View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Elsevier - Publisher Connector

ELSEVIER
Artificial Intelligence 114 (1999) 239–255
Artificial Intelligence
www.elsevier.com/locate/artint

Automating planning and scheduling of shuttle payload operations

S. Chien *, G. Rabideau [1], J. Willis [2], T. Mann [3]

Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, MS 126-347, Pasadena, CA 91109-8099, USA

Received 24 July 1998

Abstract

This paper describes the DATA-CHASER Automated Planner/Scheduler (DCAPS) system for automated generation and repair of command sequences for the DATA-CHASER shuttle payload. DCAPS uses general Artificial Intelligence (AI) heuristic search techniques, including an iterative repair framework in which the system iteratively resolves conflicts with the state, resource, and temporal constraints of the payload activities. DCAPS was used in the operations of the shuttle payload for the STS-85 shuttle flight in August 1997 and enabled a 80% reduction in mission operations effort and a 40% increase in science return. © 1999 Published by Elsevier B.V. All rights reserved.

Keywords: Scheduling; Planning; Iterative repair; Space operations; Local search; Autonomous spacecraft

1. Problem description

Generating command sequences for spacecraft operations can be a laborious process requiring a great deal of specialized knowledge. Typically, spacecraft command sets are large, with each command performing a low-level task. There are often many interactions between the commands relating to the state of the spacecraft. In addition, due to spacecraft power and weight limitations, the resources available on-board spacecraft are often scarce. These factors in combination make manual generation of command sequences a difficult

* Corresponding author. Email: steve.chien.@jpl.nasa.gov.
[1] Email: gregg.rabideau@jpl.nasa.gov.
[2] This work was performed while this author was affiliated with Colorado Space Grant College, University of Colorado. Email: jason.willis@jpl.nasa.gov.
[3] Email: tobias.mann@jpl.nasa.gov.

0004-3702/99/$ – see front matter © 1999 Published by Elsevier B.V. All rights reserved.
PII: S0004-3702(99)00069-7

process. Because of the importance and expense of this process, tools to assist in planning and scheduling spacecraft activities are critical to reducing the effort (and hence cost) of mission operations.

This paper describes a general system that uses Artificial Intelligence Planning and Scheduling technology to automatically generate command sequences for the DATA-CHASER shuttle payload operations. The DATA-CHASER Automated Planner/Scheduler (DCAPS) architecture presented supports direct, interactive commanding, rescheduling and repair, resource allocation, and constraint maintenance.

DCAPS implements search algorithms for two problems: initial schedule generation, and schedule repair/refinement. In initial schedule generation, DCAPS generates a default schedule to perform science observations from the null schedule (i.e., an empty schedule). DCAPS supports domain specific and randomized initial schedule generation strategies. In schedule repair/refinement, DCAPS accepts an existing schedule with conflicts (i.e., resource oversubscriptions, state conflicts, etc.) and performs modifications to make the schedule consistent with the spacecraft constraints. DCAPS implements this functionality by using "iterative repair" search techniques (e.g., [16]). Basically, this technique iteratively selects a schedule conflict and performs some action in an attempt to resolve the conflict. In iterative repair mode, DCAPS is naturally well adapted for human interaction. In this mode, a user can move, add and delete activities in order to alter the schedule to their preferences. DCAPS can then be invoked to repair state, resource, and temporal constraint conflicts caused by these modifications. Using an automated planner/scheduler (e.g., DCAPS) in this fashion, command sequence generation can be performed by scientists who need not be spacecraft and sequence engineer experts. This allows the scientist to become directly involved in the command sequencing process. Additionally, if there are changes in the spacecraft state (e.g., faults) or user-defined goals (e.g., science opportunities), the repair algorithm allows simple rescheduling that attempts to minimize disruption of the original schedule. Finally, the highly restrictive payload resources and constraints are constantly monitored and conflicts automatically avoided.

The sequence generation problem addressed by DCAPS includes both planning and scheduling according to typical Artificial Intelligence definitions. DCAPS performs planning in that it determines appropriate actions to achieve state and resource values required to achieve goals. It performs scheduling in that these selected activities must be temporally placed to satisfy (aggregate) resource, state, and timing constraints required by the operations constraints.

The DCAPS system was developed for operation of the DATA-CHASER shuttle payload, which was developed and managed by students and faculty of the University of Colorado at Boulder. DATA-CHASER is a science payload, with a primary focus on solar observation. The main activities for the payload involve science instrument observations, data storage, communication, and control of the power subsystem. Science is performed using three solar observing instruments: the Far Ultraviolet Spectrometer (FARUS), Soft X-ray and Extreme Ultraviolet Experiment (SXEE), and Lyman-alpha Solar Imaging Telescope (LASIT). These are imaging devices that operate at various spectra.

The payload resources include power, tape storage, local memory, the three instruments, and the communication bus. DATA-CHASER is also constrained by externally driven states such as the shuttle orientation and external events such as shuttle waste material venting, which affect when certain science activities can be scheduled. Payload activities must be sequenced while avoiding or resolving conflicts. The DCAPS system models all of these states and resources, as well as the state and resource requirements and effects of activities. This model enables automation of command generation for the DATA-CHASER payload.

The remainder of this paper is organized as follows. First, we describe the DATA-CHASER shuttle payload and mission objectives. Next, we describe how the payload is modeled. We then describe in detail the DCAPS approach to automated command sequence generation and repair. Then, we describe how DCAPS fits in to the overall flight and ground system architecture for the DATA-CHASER mission. We then describe the experience and results from the use of DCAPS during the STS-85 flight. Finally, we discuss related work and conclusions.

2. DATA-CHASER payload

DATA-CHASER consists of two synergistic projects, DATA and CHASER, which flew as a Hitchhiker (HH) payload aboard STS-85 on the International Extreme Ultraviolet Hitchhiker Bridge (IEH-2) in August 1997 [5]. A technology experiment, DATA (Distribution and Automation Technology Advancement) demonstrated advanced semi-autonomous, supervisory operations. CHASER (Colorado Hitchhiker and Student Experiment of Solar Radiation) was a solar science experiment that served to test DATA. The DATA technologies support cooperative operations distributed between different geographic sites as well as between humans and machines, on-board autonomy, human control, and ground automation.

CHASER consists of three co-aligned instruments that take data in the far and extreme ultraviolet wavelengths. The first and oldest of these instruments (17 years old) is FARUS, which takes a continuous spectrum from 115 nm to 190 nm with a resolution of 0.12 nm. LASIT takes images of the full solar disk of the sun in the Lyman-alpha wavelength (121.6 nm) with a Charge Injected Device imager. The final instrument in the scientific package, SXEE, consists of four photometers, each having a different metallic coating so as to enable them to look at different wavelengths between 1 and 40 nm. The objective of these instruments is to measure the full disk solar ultraviolet irradiance and obtain images of the sun in the Lyman-alpha wavelength, providing a correlation between solar activity and radiation flux as well as an association of Lyman-alpha fluxes with individual active regions of the sun.

The flight segment of the DATA-CHASER project consists of a canister that is equipped with a Hitchhiker Motorized Door Assembly (HMDA), which houses the instruments and their support electronics. The second canister contains the flight computer for the payload as well as the 2 GB Digital Audio Tape (DAT) drive that is used to store all data that is collected during the mission. The payload data is also sent to the ground system through both low rate (available 90% of the time, at 1200 bps) and medium rate (available when scheduled, at 200 kbps). The payload is also capable of receiving commands sent from the ground system when uplink is available.

During the mission, the DATA-CHASER payload operated in four different modes. Most of the time, when DATA-CHASER was powered, it was in a passive mode where it was monitoring its state and notifying the ground of any changes. During the time in the mission when the orbiter was scheduled to point the bay at the sun, the DATA-CHASER payload shifted into solar active mode where all instruments take data.

The data was both written to the DAT drive on board and downlinked to the ground system for immediate data analysis. Several times during the mission, DATA-CHASER took data while not pointing at the sun. This data was used for testing various portions of the DATA experiment with non-solar-pointing data in addition to being used for instrument calibration.

One of the consequences of flying on the shuttle system is that shuttle resources are shared and, hence, limited, and their availability is subject to change every 12 hours (the frequency at which NASA changes shuttle flight plans). These resources include access to uplink and downlink channels, and time that payload operations are allowed. In addition to these resources, any given payload may also have environmental constraints that restrict the payload state and activities when shuttle contamination events are occurring. In addition, a payload typically has thermal constraints, which would limit the duration of payload exposure to the sun (or away from the sun).

STS-85, the flight on which the DATA-CHASER payload flew, was one of the most complicated flights that the shuttle has flown to date. In addition to the DATA-CHASER payload, there were four other payloads sharing the same HH bridge. In addition to the IEH-2 bridge, the shuttle carried another HH bridge, a pallet payload, and a Spartan deployable satellite. Needless to say the shuttle pointing requirements were tight.

In addition to modeling the internal constraints and resources of the payload, DCAPS had to search the shuttle flight plan for times when the payload was allowed to operate, downlink data, uplink new command sets, and when scientific instruments had to be protected from contamination events.

DATA-CHASER was an interesting scenario for scheduling because of the complex data and power management involved in the science gathering. An automated scheduler must find an optimal “data taking” schedule, while adhering to the resource constraints. In addition, the scientists would like to perform dynamic scheduling during the mission. As an example, the summary data may indicate the presence of a solar flare. If this occurs, scientists have different requirements and goals, such as higher priorities on certain instruments or longer integration times. These new goals may require a different schedule of activities.

3. Modeling the payload

In order to use the DCAPS system, the user must write a software model of the mission activities and spacecraft resources. DCAPS uses the Plan-It2 system [7] to model the spacecraft activities and constraints, thus the model is expressed in the Plan-It2 modeling language. Modeling in the Plan-It2 language involves defining a set of objects and describing how they interact. These definitions are then used by the scheduler to create instances of the objects and reason about specific interactions (e.g., state and resource conflicts) in the schedule. The two major types of objects in the model are *activities* and *resources*.

3.1. Activities

Activities are used to model the events that affect the DATA-CHASER payload and the actions that the DATA-CHASER payload can take. All activities have certain basic components: a duration, a list of slots, and a list of slot-value assignments. The activity duration is simply a time range. Slots are parameters of activities that may represent resource usage. In addition, certain types of activities (described below) have a list of subactivities. For these activities, the user can also define a set of temporal constraints between the subactivities. Next, we describe in more detail the four basic types of activities: events, steps, step-activities, and activities.

- *Events* are used to model activities that do not occur in a fixed relation to other activities (e.g., Tracking and Data Relay Satellite System (TDRSS) contacts) and are not part of an activity hierarchy.
- *Steps* are the “leaf” nodes in the activity hierarchy tree. In other words, they do not contain any subactivities. Steps cannot be instantiated without their parents and are used to model the activities at the lowest level of detail. For instance, we model an activity called CHASER-heating, which consists of two steps, CHASER-heater-on and CHASER-heater-off.
- *Step-activities* are used to model activities at a middle level of abstraction. They can contain subactivities, but must also have parent activities. In DCAPS, we model an activity SXEE-Data-Take, which models the SXEE instrument opening its aperture and taking a scan. In this case, there is a step-activity called SXEE-Scan-Step, which has four sensor read steps and cannot be instantiated by itself.
- *Activities* are used to model activities at the highest level of abstraction. They are the “root” nodes in the hierarchy tree, containing subactivities, but no parent activity. An abstract activity inherits all attributes of its predecessors. When an abstract activity is detailed, it is replaced by its subactivities—showing events and resource usage at a finer granularity. The activity and event objects can be instantiated by the scheduler, and the scheduler can use methods to access the varying levels of abstraction.

3.2. Resources

Resources define the various physical resources and the constraints they impose. Resources come in essentially five varieties: state, concurrency, depletable, non-depletable, and simple.

- State resources are used to model the systems in the DATA-CHASER payload that have states associated with them. For each state resource, the modeler must specify the possible values that the state can be. Most of the systems have at least one state variable, which is whether or not they are activated. Shuttle orientation is also modeled as a state variable.
- Concurrency resource constraints are used to model rules that stipulate that an activity either must occur during another activity or cannot occur at the same time as

another activity. One relationship that is modeled with a concurrency resource is the requirement that a downlink or uplink can only occur during contact with a TDRSS satellite. This is modeled as a resource that is present when there is TDRSS contact activity and required when there is a downlink or uplink activity.

- Depletable resources are used to model resources with a fixed quantity, such a fuel or RAM. Activities can use some finite amount of a depletable resource, which may or may not be restorable. The amount used by the activity is persistent to the end of the schedule. In addition, the modeler must specify a maximum capacity for each depletable resource. In DCAPS, RAM is modeled as a depletable resource. Science observations produce data and use some amount of the depletable resource. Other activities, such as a transfer to permanent storage, may restore this resource.
- Non-depletable resources are used to model resources with a limit to the usage at any one time, but are reset at the end of the activity that consumes the resource. Similar to depletable resources, nondepletables are assigned a maximum capacity. Resources like power are modeled with non-depletable resources.
- Simple resources are used to model devices that can only be used by one activity at a time. For instance, each of the instruments on board DATA-CHASER, FARUS, SXEE, and LASIT, are capable of taking only one image at a time and are modeled with simple resources.

The DATA-CHASER model is fairly large, containing 67 resources and 58 activity types. The payload required 7 resources to model the impact of exogenous events such as shuttle contamination events, day/night cycles, shuttle maneuvers, and other external activities which impact payload operations. The payload also required 6 resources to represent possible failed states for major instruments/subsystems. For each instrument, a number of resources would be required. For example, for SXEE, there are 6 resources. One resource represents the instrument itself. Two resources are required to represent the instrument door—one for the open/closed state and another for the closing process which draws power. A SXEE-failure resource represents if the instrument is known to have failed (and hence disables the scheduler from scheduling any SXEE activities). A SXEE-power resource tracks the power consumption of the SXEE instrument, and a SXEE-relay resource models the hardware relay used to enable/disable the SXEE instrument. In addition to the instruments, there are a number of system-wide resources to be tracked by DCAPS. Total power consumption and energy usage (for thermal considerations) are tracked for each canister. Finally, science and engineering data must be processed through a set of 3 buffers onboard the spacecraft as well as the secondary storage DAT tape drive.

The DATA-CHASER model also contained a significant number of activity types (58). Of these the vast majority (25) were hardware control commands. Fourteen commands related to acquisition of science and engineering data, and 6 commands controlled the downlink capability (TDRSS, medium-rate, and low-rate). Nine (9) commands were used to represent possible subsystem failures (e.g., to disable use of certain instruments and subsystems). Seven (7) activities were used to represent exogenous events (such as medium rate downlink coverage, solar pointing for the shuttle bay, etc. The activities can also be viewed from an instrument centric perspective. From this viewpoint, the SXEE instrument has commands to: transfer SXEE data from the SXEE instrument to the general instrument buffer, to open and close the SXEE instrument door, to control the SXEE-relay which enables use of the instrument, to take a data scan, to take a dark scan (with the instrument door closed for calibration purposes), and two macro-commands which each perform several of the typical steps to take and transfer an image to storage.

4. The DCAPS automated planner/scheduler

The DATA-CHASER Automated Planner/Scheduler was part of the DATA-CHASER mission operations software. It was a ground-based intelligent tool used for developing a schedule of commands for uplink to the payload [15]. There are two phases of operating the DCAPS system: initial schedule generation and interactive repair phase.

During initial schedule generation, DCAPS produces a complete, valid schedule of payload operation commands from a model, initial state, and set of high-level goals. In the interactive repair phase, it takes intermediate, invalid schedules (resulting from user changes) and produces a similar, but valid schedule.

The planner/scheduler consists of two main parts, the Plan-IT II (PI2) sequencing tool [7] and the schedule reasoner. PI2 was written by William C. Eggemeyer and originally designed as an "expert assistant sequencing tool". PI2 includes a GUI that allows for easy manipulation of the schedule. In addition, it serves as an activity/resource database that supplies valuable information to the schedule reasoner. PI2 supports complex monitoring and reasoning about activities and their constraints. In order to provide these functions, PI2 requires a model of the payload in the PI2 modeling language (as described above). The schedule reasoner uses Artificial Intelligence (AI) techniques to automatically generate new schedules, repair existing faulty schedules, and optimize valid schedules. PI2 provides information about resource availability and conflicts; the scheduler must decide which activities to use to resolve the conflicts and where to place the activities temporally. Plan-IT II and the schedule reasoner work together to provide fast, easy sequencing of mission activities.

4.1. Schedule database

In the DCAPS system, PI2 is used primarily as a "schedule database" and resource constraint checker. It was originally developed as a graphical sequencing tool. Activities and resources are displayed on a graphical output. An activity represents some mission event that occurs over a period of time and uses some of the mission resources. A resource represents some limited available material whose usage is modeled as discrete blocks over time.

For each type of activity and resource, PI2 displays a timeline, which represents the behavior of that activity/resource type over a period of time. When activities are created, they are placed at a specified time on the timeline. Resources used by that activity are updated to reflect the additional usage. In addition to schedule visualization, PI2 provides an easy-to-use interface for modifying the schedule. Moving activities is as simple as a click-and-drag with a mouse.

PI2 helps ease the burden on sequencers by continually monitoring all activities in the sequence. As activities are added or moved, the change in resource usage is automatically updated, and the new resource profiles are displayed. With this information available, the user can immediately see the effects of a schedule change on the mission resources. For each resource, PI2 also monitors any conflicts that are occurring on the resource.

Conflicts are time intervals where the limitations of the resource have been exceeded. These conflict intervals are highlighted in red to flag their existence for easy identification. Finally, PI2 monitors any dependencies that have been defined between activities and resources. The values of specific parameters of activities and resources may be functionally dependent on values of other parameters. PI2 automatically keeps these parameter values consistent.

PI2 also helps out by serving as an activity and resource database, producing/accepting information to/from a sequencer. The functional interface to PI2 has been extended to better assist an automated sequencer. A basic set of "fetch" functions has been developed to quickly retrieve information about conflicts and the resources and activities involved in the conflict. For example, an interface function has been written to fetch the legal times where an activity can occur in the schedule. Here, "legal times" refers to positions where no conflicts are caused by any of the resources used by the given activity.

In addition to fetching information about the current state of the schedule, the user will need to be able to change the current state in attempt to fix or optimize the schedule. Some basic primitive functions are provided by PI2 to allow an external system to add and move activities, change their duration, etc. These primitives make up the set of actions that a scheduler can take when trying to resolve conflicts.

4.2. Schedule reasoner

The second major component of DCAPS is the automated schedule reasoner. The schedule reasoner provides three capabilities: initial schedule generation, schedule repair, and schedule optimization. In initial schedule generation, a schedule is generated from a set of user requested activities. In schedule repair, the scheduler will automatically restore the consistency of the sequence after arbitrary user interaction by rescheduling using repair actions. The scheduler repairer iteratively attempts to resolve each conflict, which involves making choices on what to repair and how to repair it. In a more advanced extension to iterative repair, schedule optimization can be performed. In this technique, portions of the schedule are examined and possibly rescheduled to improve a portion of the schedule which may not contain conflicts (we omit description of optimization due to space constraints).

Initial schedule generator

The first step in sequencing spacecraft commands is to come up with an initial schedule of events for each phase of the mission. This process has been partially automated in DCAPS with the schedule generator. Expressing schedules and partial schedules to be generated is done through user defined goals. There are two ways in which user goals are handled in DCAPS. First, initial science and engineering goals are handled with parameterized scheduling functions. Each function implements a goal. For example, there is a "Place-Power" function that schedules power switching activities in appropriate places

```
buildInitialSchedule()
  turn on the two canisters 2 hours into the mission
    and leave them on for the duration (cmds DataRelayOn, ChaserRelayOn)

  for each interval in which the shuttle is pointing at the sun and there is not a contamination event
    open the hitchhiker door (HMDAOpen) at the start of the interval
    close the hitchhiker door (HMDAClose) at the end of the interval

  for each interval in which the shuttle is not in a solar pointing state for longer than 30 minutes
    turn on the canister heater 30 minutes after the solar pointing
    turn off the canister heater at the beginning of the next solar pointing interval

  ;; place farus, sxee, lasit data-takes
  loop until no legal times for a data-take
    find legal times for data-take (during the solar pointing non contamination events)
    place data-take at earliest possible start time
    place a DAT transfer after the data-take
```

Fig. 1. Building an initial schedule.

based on some engineering parameters. Parameters may include such things as a minimum time between switching, or a power on during a particular state of a different resource.

Second, science goals can also be expressed through data-take requests, which do not have to be a part of the initial schedule generation. For example, a scientist can request ten additional scans from a particular instrument to occur any time during some phase of the mission. This type of general request does not include specific locations or necessary supporting activities. The scheduler will simply place them at random positions and allow any conflicts to be resolved by the automated repairer.

DCAPS supports two modes for automated initial schedule generation: a domain specific schedule generation algorithm and a randomized scheduling algorithm. The domain specific scheduling algorithm is shown in Fig. 1. In this approach, the scheduler inserts necessary setup activities (powering on the payload and controlling the payload doors) and schedules an even mix of observations by sweeping forward in time. However, little effort was devoted towards optimizing this approach.

DCAPS also supports a randomized initial schedule generation algorithm. In this approach, the scheduler merely uses random placement to attempt to place science observations. As expected, the random approach performs significantly worse than the domain specific approach.

Schedule repairer

The generated initial schedule may still violate some of the spacecraft constraints. Also, the scientists and engineers might feel that their goals were not completely satisfied or that they could be better achieved by an alternate plan. In these cases the users want to be able to interact with and modify the generated schedule. These modifications may introduce new conflicts into the schedule. The schedule repair capability can automatically repair these introduced conflicts, freeing the user from this burden and reducing overall mission

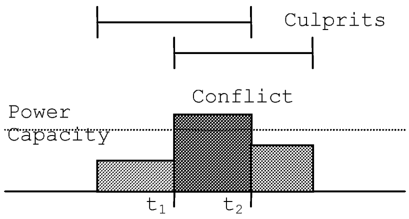

Fig. 2. Conflicts.

operations effort. Additionally, freeing the user of the burden of low-level repair, allows the user to spend more time modifying the schedule—allowing the combined user/software system to explore more of the space of possible solutions.

Before describing the schedule repairer, we must present a few definitions. A "conflict", is a violation of one of the resource constraints. A conflict occurs over a certain time period and is caused by activities called "culprits". For example, if the power capacity is exceeded from time t_1 to time t_2, then a conflict exists from time t_1 to time t_2, and the culprits are any activities that use power during this time (see Fig. 2).

There are three possible actions to take in attempt to resolve a conflict: move, add, or delete an activity. The "move" action involves moving one of the culprits of the conflict to a position that will either resolve the conflict or at least ensure that the moved activity is no longer a culprit. Some conflicts can be resolved by adding a new activity. These activities usually provide some resource that was previously not available. Finally, a conflict can also be resolved by simply deleting the culprits. This is obviously not a preferred method and is only used as a last resort.

The resolution of a conflict greatly depends on the type of resource that is in violation. There are five different types of conflicts corresponding to the five types of resources. A *state conflict* occurs when an activity requires the resource to be in a state other than its current state. The culprits in this type of conflict are all of the activities that require the incorrect state and the activity that changed the resource to the incorrect state. Several possibilities for resolving a state conflict include moving the culprits to another interval where the required state is present or adding an activity that will change the state of the resource to the required state.

A *concurrency conflict* is when an activity requires the presence of the resource during a time for which it is absent. The culprits in this type of conflict are all of the activities that require the presence of the resource. To resolve a concurrency conflict, the scheduler can move the culprits to an interval where the resource is present or add an activity that provides the presence of the resource.

A *depletable conflict* means that the activities of the schedule have used too much of the resource. In this type of conflict, the culprits are all the activities prior to the overflow. Some depletable resources have "resetter" activities and this sort of conflict can be resolved by adding an activity that "resets" the available resource. For example, a downlink activity will free up space in the downlink buffer.

A *non-depletable conflict* is when activities overuse a resource during a particular time interval. The culprits in this type of conflict are all of the activities that use the resource

```
ResolveConflicts (max_iterations)
  iterations:= 1
  Loop while (conflicts ≠ {} &&
              iterations ⩽ max_iterations)
    select a conflict
    select a method for resolving a conflict
      (move, add, delete)
    case move
      select culprit to move
      select time to move culprit
      move the selected culprit
    case add
      select activity type to add
      select start time for the new activity
      add a new activity instance of the selected type
    case delete
      select activity to delete
      delete the selected activity
    if no_progress then UndoLastAction()
    else fetch new set of conflicts
    iterations:= iterations + 1
```

Fig. 3. Iterative repair algorithm.

during the conflict interval. This sort of conflict can be resolved by moving or deleting culprits. There are no activities in the DATA-CHASER model that can add to a non-depletable resource.

Simple conflicts occur when two or more activities use the same simple resource at the same time. This type of conflict can only be resolved by moving culprits.

Given an initial schedule, the schedule repairer must find the correct activities to move, add, or delete and position them temporally in such a way that no conflicts remain. The scheduler relies on some interface functions to PI2 that describe the conflicts in the current schedule, describe the activities that could resolve a conflict, and manipulate the schedule. The schedule repair algorithm is an iterative loop over the conflicts in the schedule (see Fig. 3). First, the repairer must select a conflict to attack. Next, a method for resolving the conflict is chosen. Depending on the conflict type, there may be up to three methods for attacking the conflict: move, add, and delete. If "move" is chosen, then a culprit must be picked from the list of culprits in the conflict. A duration and start time are chosen for the culprit, and the culprit is moved to the new location. If "add" is the chosen method, then the repairer must decide which activity type to instantiate. Again, a duration and start time must be chosen for the new activity, and the activity is inserted at the chosen time. If the repairer chooses to "delete" an activity, then it simply must choose an activity to delete, and delete it. After the chosen action is performed, the schedule repairer checks to see if progress was made. We define progress as either decreasing the number of conflicts, decreasing the number of culprits, or decreasing the duration of the conflicts. If the action did not

Table 1
Heuristics implemented for each choice point

Choice point	Heuristics
Conflict selection	Highest priority (determined by resource)
	Highest contention (oversubscription)
	Most culprits
	Largest duration
	Least culprits*
	Smallest duration
Operation selection (move, add, or delete)	Random
Activity selection for Move operations	Move culprit which contributes most to conflict*
	Move culprit with least temporal flexibility
	Move lowest priority
Activity selection for Add operations	Add activity that reduces conflict most*
	Add the activity which has the fewest legal times
	Add highest priority
Activity selection for Delete operations	Delete culprit that contributes most to conflict
	Delete the culprit which participates in most conflicts*
	Delete lowest priority
Time selection	Choose latest start time
	Choose earliest start time
	Choose latest start time for state conflicts and earliest start time for resource conflicts*

succeed in resolving the conflict, or progress was not made, then the action is “undone”. Otherwise, the new set of conflicts are found, and the loop counter is incremented. This process continues until all conflicts are resolved, or the loop counter exceeds a user-defined maximum bound. For every choice point in the algorithm, where a selection must be made from a list of possibilities, the schedule repairer is allowed to backtrack to that point. What this means is, that if a particular choice fails, the schedule repairer may choose another from the list before giving up. If all choices fail, then a previous decision must have been incorrect, and the repairer can backtrack to the preceding choice point. All choice points, including the decision on whether or not to backtrack, are heuristic decisions and may be customized to a particular domain.

Thus, there are several choice points in the repair algorithm that are relevant for heuristic guidance:

(1) conflict selection,
(2) selection of move, add, or delete,
(3) selection of an activity with which to move add or delete, and
(4) temporal placement of the moved or added activity.

In Table 1, we outline the heuristics implemented within DCAPS. For each choice point, we highlight the heuristic method used for the DATA-CHASER mission with an asterisk.

Schedule optimization

Often there may be many legal schedules, all of which are not equally preferred by the users. In the extreme case, the empty schedule (e.g., do nothing, or some schedule of this form) is usually a legal schedule. In the DATA-CHASER mission, the dominant quality measure is science return—which can be roughly measured by the number of science measurements taken and downlinked in the current planning cycle. In order to improve the quality of the DCAPS-produced schedules, we implemented a simple schedule optimization algorithm that accepts as input an oversubscribed schedule. This algorithm first expands all of the activities in to the lowest level (because the most detailed resource modeling may allow a more densely packed schedule). The algorithm then performs a forward sweep through the schedule in which each activity is moved to its earliest start time. This has the effect of packing the activities towards the start of the schedule potentially opening room for the extra activities towards the end. In the DATA-CHASER case, the oversubscribed activities are science data-takes and the oversubscription is due to over-use of the instrument, communications bus, and buffer resources. The schedule optimization algorithm takes an oversubscribed schedule and packs in the science observations more closely—thus allowing further science observations to fit into the schedule. This optimization algorithm can be viewed as a simplified version of the schedule packing described in [1] and the doubleback algorithm described in [4].

5. Application development, deployment, and system integration

DCAPS was developed by the JPL Artificial Intelligence group as part of a set of early prototypes of automated planning and scheduling engines for use by NASA's New Millennium Program. Later, when the DATA-CHASER mission operations automation problem was studied, we determined that the iterative repair capabilities of DCAPS would be well suited for mixed-initiative partially automated, human in the loop, shuttle payload operations. At this time DCAPS was modified to meet a number of minor user interface requirements and the DATA-CHASER model was constructed over a series of software spirals with each model increasing in coverage and fidelity. The total JPL AI Group effort involved in the development of DCAPS and initial modeling was approximately 1.4 work-year. The total effort by CSGC to deploy the DCAPS system was on the order 0.4 work-years.

DCAPS was integrated into the End-to-End Mission Operations System (EEMOS) used for the DATA-CHASER portion of the STS-85 payload. This EEMOS architecture is also being evaluated as part of the Fire and Ice pre-project [11]. The DATA-CHASER EEMOS consisted of seven parts: Command and Control, Fault/Event Detection Interaction Reaction (F/EDIR), DATA/IO (Data handling), the Ground Database, the Graphical User Interface, the software testbed, and finally the planning and scheduling system (DCAPS).

The command and control language used, System Command Language (SCL, also known as Spacecraft Command Language), integrates procedural programming with a real-time, forward-chaining, rule-based system. DCAPS interfaces with SCL through DATA/IO by sending script scheduling commands to be scheduled either on the flight or ground system. This interface is implemented by mapping PI2 activities to SCL scripts that were written prior to flight and can be scheduled or event-triggered by activating rules. A list of these scheduling and rule activation commands is then sent to DATA/IO which forwards the list to the SCL Compiler. Once compiled, the list is sent to the payload through the next available uplink.

DCAPS is also interfaced with the ground EEMOS database, O2. O2 is an object-oriented database used to store all mission data and telemetry that is downlinked by the payload. O2 also stores a command history. Through DATA/IO, DCAPS requests current payload status data in the form of sensor values in the telemetry history. It also requests lists of all commands uplinked during a given time interval. These are used by DCAPS to infer command completion status as well as to get the current state of the payload so that a new schedule can be created.

During mission operations, approximately every six hours, DCAPS was asked by an operator to generate script scheduling commands and rule activations for the next six hours according to its schedule. Once this list was generated, it was reviewed by the Mission Operations staff on duty. When judged to be correct, scheduling and rule activation commands would be sent to DATA/IO during the next available uplink window.

If during that six hour period there was a major change in the NASA activities, the operations staff could use DCAPS to update the schedule script on-board. If so desired, DCAPS could generate an updated command list, ask the user to verify it, and send the list to DATA/IO to be uplinked.

6. Impact and results from use during STS-85

Unfortunately, difficulties were encountered during the development and integration of the real-time DATA-CHASER flight software. Due to these difficulties and hard shuttle payload delivery constraints, the real-time onboard command execution software for the payload did not have several capabilities that were originally designed. First, the onboard software was unable to command the SXEE and LASIT instruments. Second, the onboard software did not have the capability to store and execute time-tagged command loads, thus all operations had to be carefully synchronized with real-time shuttle uplink windows. Third, the onboard tape storage device (DAT) was not functional. This meant that data storage was limited to the onboard solid state buffers. However, since the LASIT instrument was the most significant producer of data by over an order of magnitude this was not a major problem. The first and second limitations described above meant that having an automated planning system to automatically coordinate the complex timing constraints was even more important; manually attempting to enforce such timing constraints would increase the chance of operator error causing a loss of data. Likewise, being able to replan quickly and automatically when shuttle activities changed (such as downlink or uplink windows and solar view periods) was also critical.

Carrying the DATA-CHASER payload, STS-85, the Space Shuttle Discovery launched 7:41AM PST on Thursday August 7, 1997. Mission operations, including mission planning and scheduling were performed for the two week flight. During the first five days of DATA-CHASER operations, DCAPS was used in manual mode. In this mode activities were placed manually and DCAPS was used to: validate constraints, identify constraint violations, and to generate the actual command files. During the last seven days of the payload operation, DCAPS was used to automatically generate schedules. In this phase the domain specific initial schedule generator was used to generate an initial schedule. Due to network lag times (DATA-CHASER was operated primarily from Colorado Space

Grant and due to machine shortages DCAPS was running at JPL) use of the iterative repair techniques were somewhat limited. However, this limitation did not significantly impact operations—in many cases minor conflicts were repaired manually.

The DCAPS automated scheduling capability significantly impacted DATA-CHASER mission operations. DCAPS enabled an 80% reduction in the amount of effort to produce operations plans. Manual generation of a 6-hour operations plan would require from 30 to 60 minutes in manual mode of operations and from 7 to 9 minutes using the DCAPS automated scheduling capability. This reduction in effort is because DCAPS can automatically generate an acceptable or near acceptable schedule very quickly. The number of modifications (if any) to make a DCAPS generated schedule acceptable can be made far faster than manually generating a schedule from scratch. DCAPS also enabled a 40% increase in science return. Manually generated plans had 2–3 instrument scans per viewing opportunity whereas DCAPS generated plans had 3–4 scans per viewing opportunity. This is because DCAPS could directly monitor and track all of the complex timing constraints involved in the instrument activities and pack activities more tightly than operators manually placing instrument activities. During this seven days of DCAPS automated use, DCAPS scheduled a total of 93 science scans and 202 payload commands.

One significant feature of the DCAPS system is its declarative representation of flight rules and spacecraft constraints. This feature was tested during the STS-85 flight in the following manner. When initial command sequences were uplinked, the flight software rejected a number of commands immediately following a reset command. This was due to the fact that the initial flight rules were constructed with the understanding that immediately following a reset, commands could be issued to the payload. Actual operations showed that a delay of 30 seconds was needed before the payload could accept commands. When this problem was noticed and isolated, it was a simple manner to quickly update the DCAPS model to require this delay so that future command sequences would execute without problem. This aspect of ease of modification is key in that spacecraft characteristics and operating procedures constantly evolve throughout the mission lifecycle as the spacecraft characteristics and mission priorities evolve.

7. Summary and related work

Iterative algorithms have been applied to a wide range of computer science problems such as traveling salesman [10] as well as Artificial Intelligence Planning [3,8,12,14]. Iterative repair algorithms have also been used for a number of scheduling systems. The GERRY/GPSS system [6,16] uses iterative repair with a global evaluation function and simulated annealing to schedule space shuttle ground processing activities. The Operations Mission Planner (OMP) [2] 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 [9] shows how the min-conflicts heuristic can be used not only for scheduling but for a wide range of constraint satisfaction problems. The OPIS system [13] 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.

In summary, DCAPS represents a significant advance from several perspectives. First, from a mission operations perspective, DCAPS is important in that it significantly reduces the amount of effort and knowledge required to generate command sequences to achieve mission operations goals. Second, from the standpoint of Artificial Intelligence applications, DCAPS represents a significant application of planning and scheduling technology to the complex, real-world problem of spacecraft commanding. In particular, significant quantitative improvements in operations efficiency were documented during the STS-85 flight. Third, from the standpoint of Artificial Intelligence research, DCAPS mixed initiative approach to initial schedule generation and iterative repair represents a novel approach to solving complex planning and scheduling problems.

Acknowledgements

This work was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The authors also gratefully acknowledge the contributions of other participants in the DCAPS and DATA-CHASER projects who also contributed to the work described in this paper: Peter Stone, Curt Eggemeyer, and Sam Siewert.

References

[1] A. Aldas, A post-process optimization algorithm for resource-constrained project scheduling, in: Working Notes International Workshop on Planning and Scheduling for Space Exploration and Science, Oxnard, CA, 1997.
[2] E. Biefeld, L. Cooper, Bottleneck identification using process chronologies, in: Proc. IJCAI-91, Sydney, Australia, 1991.
[3] S. Chien, G. DeJong, Constructing simplified plans via truth criteria approximation, in: Proc. Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL, 1994, pp. 19–24.
[4] J. Crawford, An approach to resource constrained project scheduling, in: Proc. Artificial Intelligence and Manufacturing Research Planning Workshop, Albequerque, NM, 1996. Also http://www.cirl.uoregon.edu/crawford/papers/albur.ps.
[5] DATA-CHASER documents, Annual Report.
[6] M. Deale, M. Yvanovich, D. Schnitzius, D. Kautz, M. Carpenter, M. Zweben, G. Davis, B. Daun, The space shuttle ground processing system, in: Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994.
[7] W. Eggemeyer, Plan-IT-II bible, JPL Technical Document, 1995.
[8] K. Hammond, Case-Based Planning: Viewing Planning as a Memory Task, Academic Press, San Diego, CA, 1989.
[9] M. Johnston, S. Minton, Analyzing a heuristic strategy for constraint satisfaction and scheduling, in: Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994.
[10] S. Lin, B. Kernighan, An effective heuristic for the traveling salesman problem, Oper. Res. 21 (1973).
[11] S. Siewert, E. Hansen, A distributed operations automation testbed to evaluate system support for autonomy and operator interaction protocols, in: Proc. 4th International Symposium on Space Mission Operations and Ground Data Systems, ESA, Forum der Technik, Munich, Germany, 1996.
[12] R. Simmons, Combining associational and causal reasoning to solve interpretation and planning problems, Technical Report, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1988.
[13] S. Smith, OPIS: A methodology and architecture for reactive scheduling, in: Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994.

[14] G. Sussman, A computational model of skill acquisition, Technical Report, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1973.

[15] G. Rabideau, S. Chien, T. Mann, C. Eggemeyer, P. Stone, J. Willis, DCAPS user's manual, JPL Technical Document D-13741, 1996.

[16] M. Zweben, B. Daun, E. Davis, M. Deale, Scheduling and rescheduling with iterative repair, in: Intelligent Scheduling, Morgan Kaufmann, San Francisco, CA, 1994.