Automating Mission Scheduling
for Space-Based Observatories

Barney Pell
Nicola Muscettola
Othar Hansson
Sunil Mohan

Automating Mission Scheduling for Space-Based Observatories

Nicola Muscettola
Recom Technologies
NASA Ames, MS: 269-2
Moffett Field, CA 94035-1000

Barney Pell
RIACS
NASA Ames, MS: 269-2
Moffett Field, CA 94035-1000

Othar Hansson
Heuristicrats Research, Inc.
1678 Shattuck Ave. Suite 310
Berkeley, CA 94709-1631

Sunil Mohan
Recom Technologies
NASA Ames, MS: 269-2
Moffett Field, CA 94035-1000

Abstract.
In this paper we describe the use of our planning and scheduling framework, HSTS, to reduce the complexity of science mission planning. This work is part of an overall project to enable a small team of scientists to control the operations of a spacecraft. The present process is highly labor intensive. Users (scientists and operators) rely on a non-codified understanding of the different spacecraft subsystems and of their operating constraints. They use a variety of software tools to support their decision making process. This paper considers the types of decision making that need to be supported/automated, the nature of the domain constraints and the capabilities needed to address them successfully, and the nature of external software systems with which the core planning/scheduling engine needs to interact. HSTS has been applied to science scheduling for EUVE and Cassini and is being adapted to support autonomous spacecraft operations in the New Millennium initiative.

1. Introduction

The use of spacecraft observatories has dramatically expanded the horizons of astronomy and profoundly influenced our understanding of the Universe. Operating such observatories has always posed challenges that are not usually present in the operation of ground based telescopes. Once in space, instruments are unaccessible for maintenance and repair. Therefore, safe operation of the space facility has to be insured to a very high degree. Streams of commands executed by the spacecraft must be guaranteed not to put the spacecraft in any dangerous situation (e.g., pointing an instrument directly toward the Sun or having too many subsystems on at the same time draining electric power from internal supplies). Moreover, space observatories are unique and precious resources for the scientific community. Very often the requests for observations exceed what can be possibly accomplished during the limited lifetime of a mission. Accommodating a high number of requests while satisfying the observation constraints calls for the advance generation of an efficient observation schedule.

Because of the combination of safety and efficiency issues, automating the generation of detailed sequences of spacecraft commands is a challenging computer science problem. Its solution will facilitate spacecraft operations, reduce operational costs and ultimately allow scientists to operate spacecraft more directly, with potentially high scientific payoffs.

In this paper we describe the HSTS (Heuristic Scheduling Testbed System) project (Muscettola 1994a). The project is developing technologies for automated planning and scheduling and is applying them to the automated generation of spacecraft commands. We first give an overview of science mission operations with particular emphasis on planning and scheduling issues and we look at the main cost drivers. On the basis of this discussion, we point out several requirements on practical solutions to the computational problems in this domain. We then discuss the HSTS technology and our progress to date in providing a solution which satisfies these constraints.

Mission operations have traditionally accounted for a large part (40% on average) of the cost of a mission. Examples of such activities are science scheduling, command up-link (i.e., the generation and communication to the spacecraft of "correct" command sequences), and the monitoring of spacecraft operations. Operations teams vary in number depending on the mission, but range from tens of people for a small astrophysical observatory (e.g., EUVE) to several hundreds in the case of larger missions (e.g., Voyager, Hubble Space Telescope, and Cassini).

At present, the space program is exploring ways to reduce the cost of its space missions. One avenue is to use several small spacecraft in order to achieve the goals achievable by a single large spacecraft. This approach has the advantages of reducing the complexity of spacecraft operations and reducing spacecraft development costs through the reuse of standardized hardware components. However, when the cost of software development is taken into account, this approach will not necessarily be effective. The structure and scientific tasks of these smaller spacecraft are likely to vary greatly from one mission to another. Without software reuse and cross-fertilization across missions, mission support systems will be developed separately, as has been the case with previous missions. If this approach is followed in the future, we will encounter an even bigger operations problem than at present.

One of the principal activities in mission operations is the generation of sequences of commands for the spacecraft. This involves a fairly complex process with several distinct stages. Proposals are submitted to the organization responsible for managing the mission's scientific activities. A proposal is typically organized in a program of observations all aiming to achieve a scientific objective. Each observation in a program can specify the use of one of several instruments on the spacecraft. A program can contain a diverse set of temporal constraints including precedences, windows of opportunity for groups of observations, minimum and maximum temporal separations, and coordinated parallel observations with different viewing instruments. Proposals are reviewed on the basis of their scientific merit and a set of observations is accepted and selected for execution by the spacecraft.

Several missions have adopted a process that generates a command sequence in two separate steps: long-term scheduling and short-term scheduling.

Long-term scheduling takes into account an aggregate characterization of the spacecraft's observation capacity and long term periodic viewing constraints. In the case of Earth orbiting observatories, for example, a periodic viewing constraint arises because of the change of the position of the Sun with respect to a celestial object during a year. When a celestial object falls within a given angle centered around the Sun's position, it is not possible to observe the target because this would require pointing instruments too close to the Sun and therefore endangering the spacecraft and instruments. When the object is farther away from the sun, observations are possible but not preferable because of the possibility of sunlight polluting the instrument. Taking into account such constraints, long-term scheduling algorithms (Bresina 1994; Johnston & Miller 1994) distribute the accepted proposals on the mission time-line, decompose the time-line in regular intervals (e.g., monthly or weekly) and determine which set of observations need to be executed for each interval.

Short-term scheduling takes the result of long-term scheduling, focuses on one time interval and the associated observations, and generates the sequence of commands that the spacecraft will execute to implement the observations. This requires the consideration of a much more detailed characterization of the spacecraft subsystems and operations constraints. For example, most targets observed by satellites in low-earth orbit are periodically occluded by Earth (once per orbit). Similarly, interplanetary probes must synchronize their communication to Earth during the periods of visibility of the Deep Space Network antennas (which depend on the daily rotation of the Earth). Short-term scheduling must also take into account several stringent constraints on the operations of spacecraft subsystems. These include limited available electric power, and maintenance of acceptable temperature profiles on the spacecraft structure. The pointing subsystem is responsible for orienting the spacecraft toward a target, and locking it in a well defined position of the field of view of the designated scientific instrument. Spacecraft with several instruments may not have enough available power to keep all instruments operational simultaneously. Moving an instrument between operation and quiescence may require complex reconfiguration sequences which must be coordinated among various instrument components. Reconfigurations must also be appropriately synchronized among different instruments. Data can be read from the instruments and directly communicated to Earth; more usually it is temporarily stored on an on-board memory unit and communicated to Earth at a later time.

The complexity of the short-term scheduling problem and the large variety of constraints that need to be considered from one mission to the other makes it difficult to devise totally automated solutions to the problem. So far, short-term scheduling is a process that requires various degrees of human involvement. In the development and application of the HSTS technology we aim to demonstrate that Artificial Intelligence (AI) planning and scheduling can provide the basis for a more systematic and cost-effective approach to the development of automatic and mixed initiative mission operations systems, in particular for short-term scheduling. To be successful, however, we have not approached the problem by providing a turn-key, totally automatic system that is not integrated into the current, mostly manual scheduling process. A proposed framework may not be useful (and therefore not be used) if it tries to duplicate functionalities already provided, if its problem solving and representation paradigm does not easily accommodate the information typically used by human operators, or if it does not enable a mixed-initiative problem solving process between human and computer.

The rest of this paper is structured as follows. Section 2. describes the organizational process of building a science schedule and its computational challenges. Section 3. provides a set of necessary features for a planning and scheduling tool to be practical in supporting this process. Section 4. describes the architecture of HSTS. In particular, we elaborate on desiderata for one particularly important requirement, that of an appropriate language in which to model spacecraft components and scientific objectives. This discussion is grounded in our modeling language, HSTS-DDL, for use with the HSTS integrated planning and scheduling system (Muscettola 1994a). We then discusses

the integration of external knowledge within our modeling language and scheduling tool. Section 5. discusses current and future applications of HSTS.

2. Science Schedule Construction

A fully automated science mission scheduler would take as input a prioritized set of basic science goals and constraints (such as "observe the impact of comet x on planet y" and "never point the instrument at the sun"), and would produce as output a detailed list of primitive spacecraft instructions (a *sequence*). The execution of a sequence would lead to the achievement of the science goals subject to the constraints. Sequence quality can be measured in terms of the number and importance of the science goals achieved, in terms of the efficiency of use of available resources, and in terms of the effort needed to validate that the sequence satisfies its constraints.

There is an important tradeoff in sequence generation between sequence optimality and tractability. Micro-optimized sequences (i.e., sequences optimized at the level of individual spacecraft-instructions) may achieve more science goals, but generating such optimized sequences can be very expensive. This is in fact the traditional approach which has been used, for example, in the Voyager missions. Voyager was a fly-by mission in which each period of a few months of intense active observation was separated by years of relative inactivity. During the Neptune encounter, for example, about 1000 observations were executed over a period of about 5 months. In order to integrate *observation requests* from about 50 scientists into a sequence, a team of approximately 50 *support schedulers* worked full-time for a period of 4 years in advance of the encounter. The problem was an order of magnitude larger for each of the encounters with the closer planets of Jupiter and Saturn. Voyager had 11 science instruments that could operate in parallel. Each support scheduler approached the sequencing of each observation request on each instrument as a micro-optimization problem, and further optimization was then achieved by optimizing combinations of observation-specific sequences.[1]

This extremely labor-intensive approach to sequencing is no longer viable. This is due in part to limitations on available budgets and in part to the nature and projected number of future missions. Thus, most space missions have abandoned the micro-optimization approach used in Voyager and have adopted an organizational model for science scheduling that emphasizes reuse of a small set of standardized sequence components.

Before any spacecraft sequencing is done, a set of scientists and engineers first develop a small set of programs composed out of primitive spacecraft and instrument instructions, which are intended to achieve certain common subgoals. Scientists and engineers refer to these programs as *modules*. For example,[2] one module might lock the spacecraft onto a target once the instrument is pointing in the vicinity of the target; another module might turn on the tape recorder; another module might dump the contents of a specified tape recorder and send the results back to earth; and yet another module might move the spacecraft's field of view from one location to another, an operation called *slewing*. It is important to note that the modules themselves are not necessarily fixed action sequences, but may be conditional and parameterized networks of commands. This means that substantial work may be invested to verify that calling a particular module with a given set of parameter values in a particular spacecraft state will have its intended consequences. Of course, as their name suggests, the modules are designed to be modular and, thus, relatively insensitive to external conditions.

---

[1] It should be noted that Voyager schedulers did evaluate the use of AI planning tools for this problem (Vere 1983). These tools were abandoned because of poor search performance.

[2] The specific sets of modules vary with each mission. Hence, the examples of modules and spacecraft constraints provided in this paper should be taken only as a didactic interpretation by the authors. Actual modules may refer to a considerably lower level of detail than our examples suggest.

Once a set of modules has been defined, the iterative process of distributed scheduling commences. During each iteration, scientists formulate their science *requests* (the tasks they want accomplished) in terms of the modules they would like performed. A particular request may specify all of the parameters of each module, or may specify some parameters and place constraints on other parameters. For example, the particular time at which the module is executed may not matter so long as that time corresponds to night at the receiving earth station or so long as a particular planet is visible to the instrument at that time. Similarly, the requester may only care that *some* tape recorder is used to store information, in which case she would specify the `record-instrument` module without specifying its tape-recorder parameter.

Given a set of module requests and a history of the outcomes of the previous iterations, the task of the *schedule coordinator* is to produce a schedule for the next iteration which satisfies a set of generally informal criteria. This involves characterizing the various requests in terms of their priority and flexibility, adding highly constrained or constraining tasks which have high priority to the schedule, and asking users to negotiate on requests which may be difficult to achieve as stated. A particularly difficult computational task in this process is that of *schedule refinement*, or converting the partially constrained modules requested by scientists into a more detailed schedule which satisfies the constraints on these requests. This refinement takes place by the following two types of operations:

- **constrain** some parameters in existing modules. These constraints may be local to the module (e.g., constraining the absolute start time of the module) or may be relative to other modules (e.g., constraining one module to start after another has finished).

- **add** new modules to satisfy the preconditions of existing modules. These added modules essentially "glue" different modules together. An example is to add a slewing module between two observation modules. Like the original modules, these added modules need not specify all parameters.

This computational problem is similar in nature to problems treated by standard AI planning and scheduling algorithms, but with some important differences. We believe that this problem is most accurately characterized as *planning in service of scheduling*, in which both planning and scheduling are necessary but in which the major computational difficulty lies in scheduling.

In traditional precondition-achievement planning (Fikes, Hart, & Nilsson 1972), the planner is presented with a set of conditions to achieve, and the task is to find a set of actions to achieve these conditions. In the present context, human scientists do the high-level planning to convert goals into modules, and the task of the program is to refine those modules and add new ones to ensure that the overall schedule is *operational*. In this framework, the precondition-achievement left to the algorithm (to achieve conditions on modules already requested) is a very simple problem, since there are only a small number of ways in which different modules can be combined.

As discussed above, a common form of constraint on a module's parameters will be to constrain relative and absolute times of actions. While this is a key aspect of scheduling problems, this problem also necessitates extensions to traditional scheduling. In particular, most scheduling algorithms (Smith 1994) take as input a set of actions and constraints about the interactions between these actions, and attempt to assign (possibly relative) times to those actions so that the constraints are satisfied. In the current problem, in addition to time of actions, the problem-solver can also alter parameters associated with actions which may take on continuous values and affect state. Another difference is that traditional schedulers do not generate actions, as is necessary in the current problem.

3. Supporting mission operations

In our discussion of mission scheduling so far, we have focused on *computationally* difficult aspects of problem solving (e.g., producing a schedule satisfying a set of constraints). While these aspects must

be addressed by a scheduling program, it is equally important to automate or support the whole spectrum of *practical* but important functions performed by real human schedulers. This section outlines the actual process of scheduling as it relates to mission operations more broadly, with emphasis on the types of support needed from planning and scheduling tools, beyond combinatorial problem-solving.

Scheduling in practice involves making complex tradeoffs among often conflicting requirements of different machines and individuals. Human schedulers spend much of their time *explaining* why they made their decisions, and *negotiating* compromises when several tasks cannot all be achieved at once. For example, a scheduler might inform a scientist who requested an observation over a certain time period that her request might cause an instrument to be damaged, or ask her to shorten the requested time period to accommodate an important observation by another scientist.

Once a workable and agreeable initial schedule is produced, it is distributed to the individuals involved in the mission, who then use the schedule as a basis for their own subsequent planning. From this point, the human scheduler's task is to handle new considerations as they arise. The ability to make *incremental and localized modifications* to the existing schedule is crucial. The knowledge that the schedule will be broadly correct enables individuals to use an intermediate schedule as a basis for planning. Non-local changes impact the whole organization, and their cost increases the longer the schedule has been out. Local repair techniques (Minton *et al.* 1992; Zweben *et al.* 1993) may provide a useful basis for low-cost modifications to schedules. Structural domain decomposition provided by localized search techniques (Lansky 1994), may also produce local modifications during re-planning.

This process of iterative scheduling and refinement continues until final commitments are made, which are then *translated* into detailed command sequences. Much effort is spent in this translation process, and in *verifying* that the sequences are safe and satisfy the science schedule.

Even once the sequences are then up-linked to the spacecraft, the job of human schedulers continues in the case of *error recovery* – the case where things did not all go as planned. Here again, it is important to be able to explain what was supposed to be happening and to make hopefully small modifications to the schedule in light of the new circumstances. This functional requirement may overlap that discussed earlier with respect to iterative scheduling, but in this context there is a much higher emphasis on real-time recovery.

In all of these scheduling functions, *robustness* and *flexibility* are desirable properties of a schedule. Inflexible schedules increase the chance that incorporating new considerations will require global changes and increase the likelihood that uncertainty in execution will result in failure (Drummond, Bresina, & Swanson 1994; Muscettola 1994b; Hanks & McDermott 1994).

The process described in the previous section and the desiderata illustrated in this section raise strong requirements on three aspects of any automated solution:

- **language:** A domain-representation language must facilitate the representation of complex, parameterized constraints. These constraints may be based on the function and structure of spacecraft components, which suggests that the language will have to provide sophisticated modeling tools and enable domain models to be developed at multiple levels of abstraction.

- **algorithm:** A planning/scheduling algorithm must be able to operate on constraints and models expressed in such a language. It must also integrate features of planning systems (such as action generation, task reduction, and parameter specification) with features of scheduling algorithms (such as task ordering, metric and resource constraints). In addition, the algorithm must be iterative, interactive and incremental, so that its output at each stage of schedule refinement can be used as a basis for the human actions which are taken concurrently with the scheduling process.

- **external knowledge:** Lastly, both the language and algorithm must enable the user to make use of extensive knowledge stored in other software packages, as it will in practice be impossible to replicate this knowledge. As a simple example, the system must be able to figure
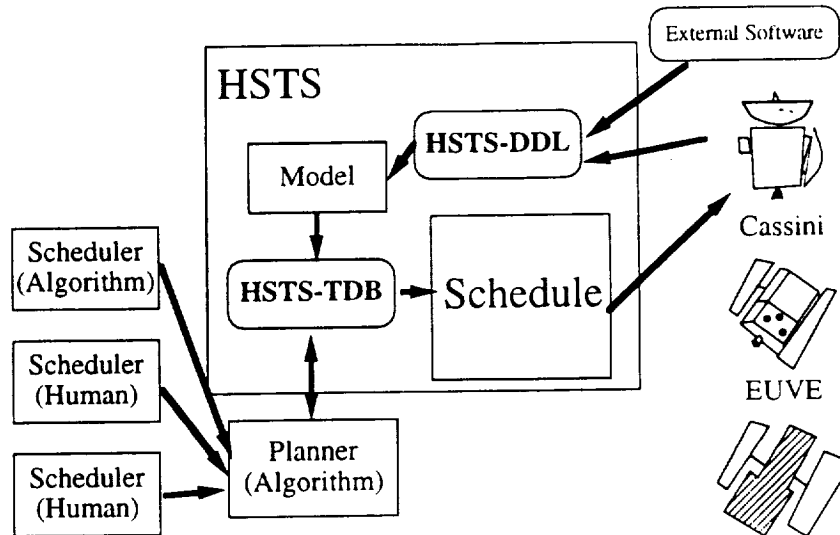
Figure 1. Block diagram architecture of HSTS.

out where Pluto will be at any given time in order to interpret the constraint "instrument pointing at Pluto."

We are currently extending the HSTS system to apply it to this domain. The HSTS scheduler embodies a design philosophy which integrates planning and scheduling, and it uses a representation language (HSTS-DDL) which facilitates the development of parameterized domain models, modular structure via an object-oriented representation scheme, and constraint representation at multiple levels of abstraction. In the next section, we discuss this system in more detail.

4. The architecture of HSTS

Figure 1 shows a block diagram for the architecture of HSTS. The two core components are the Domain Description Language (HSTS-DDL) and the Temporal Data Base (HSTS-TDB). HSTS-DDL allows the specification of a model of the domain. In a spacecraft sequencing domain, for example, a model will contain a characterization of all the constraints discussed in Section 1. HSTS-TDB supports the generation of short-term schedules that are consistent with the HSTS-DDL model. By posting assertions and constraints among assertions in the data base, a planner/scheduler sets goals, builds activity networks, commits to the achievement of intermediate states, and synchronizes system components. The tight connection between the entities that can be specified in HSTS-DDL and those that can be represented in HSTS-TDB has two main benefits. First, it assures compliance to the model. Second, it provides a strong basis for efficient schedule generation by taking advantage of the domain structure during problem solving.

We have already built several scheduling systems for short-term mission scheduling using HSTS. Their general structure involves a phase during which observations are allocated in sequence for execution on the different spacecraft instruments (scheduling) and a second phase during which detailed constraints are enforced and auxiliary activities are generated as needed (planning). To generate a short term schedule a problem solver repeatedly alternates between planning and scheduling. Currently we are using a completely automated planning phase. The scheduling phase is different in different applications and can involve human decision making. Both planning and scheduling phases are customized with heuristics depending on the domain model expressed in HSTS-DDL.

We will now focus on two aspects of HSTS: expressing domain models in HSTS-DDL and integrating HSTS with external software.

4.1. Modeling space mission domains

Accurate models of the spacecraft are essential for the generation and verification of a detailed sequence over an extended time horizon (potentially covering the entire mission duration). A less model-intensive and more reactive planning/execution approach is not feasible since the current generation of spacecraft has a very limited amount of autonomy to react to and recover from anomalous situations. We are building detailed spacecraft domain models for science mission planning by using and extending HSTS-DDL. This domain description language has many features that we believe to be fundamental for the success of an automated planning tool in this domain.

HSTS-DDL follows an object-oriented approach to domain specification. The model of a system is built by assembling models of subsystems. The state of each subsystem can be described by the values assumed by a few state variables over continuous time. This domain structuring technique, also employed by Lansky (1988), enables clearer indexing of the instantaneous state of the system when compared to the relatively unstructured state descriptions used in classical approaches to AI planning.

As an example of state variable declarations, a spacecraft behavior can be described by giving the direction where the spacecraft is pointing and the state of each of its instruments. In HSTS-DDL this can be expressed by the following class declarations.

```
(Define_Object_Class Spacecraft
  :state_variables
  ((Controllable Pointing_Status)))

(Define_Object_Class Instrument
  :state_variables
  ((Controllable Op_State)))
```

Instances of these classes can be created to describe different spacecraft (e.g., Cassini) and different instruments on a spacecraft (e.g., VIMS and UVIS on Cassini). An HSTS-DDL model is therefore highly *modular*. Models of increasing complexity can be developed by reusing and assembling smaller models. Likewise, planning/scheduling heuristics can be indexed with respect to the subsystem to which they are relevant. This facilitates both scaling and reuse.

As the previous example shows, state variables (e.g., Pointing_Status and Op_State) have an associated type (Controllable) which determines the way they will be implemented. State variables whose value can be altered by a planner/scheduler are Controllable. Their implementation allows the full range of constraint posting and propagation provided by the HSTS temporal data base. Time-lines of fixed events (e.g., when a certain star rises and sets for an earth-orbiting observatory) cannot be manipulated by a planner/scheduler. HSTS provides a reduced (and more efficient) implementation when such a state variable is declared Uncontrollable.

Beyond what is illustrated in the above example, strong *data typing* is pervasive to an HSTS-DDL model. Besides enabling increased efficiency in the implementation (an important factor for a large, practical planner/scheduler), data typing facilitates the construction of large models. A standard benefit of typed programming languages is that many errors can be detected at compile time rather than during problem solving. In HSTS-DDL type checking is implemented by a *back-end independent translator* that can translate models into data structures interpretable by a planning/scheduling system. Currently, the HSTS-DDL translator creates Lisp code compatible with the current version of the HSTS temporal data base. A future version will be geared toward a C++ implementation. Conceivably, back-end translations could be tailored for different planning/scheduling systems with semantics comparable to HSTS-DDL.

Each state variable's value typically has a finite duration. As mentioned in Section 2., the scheduling aspect of the problem is central in this domain and cannot be treated as subordinate to the planning aspect, as is often done in the AI planning community. HSTS-DDL explicitly represents

metric time and *finite capacity* associated with model entities. The value of a state variable evolves over continuous time. State variables generalize scheduling resources by combining capacity with state information.

Another powerful programming technique HSTS-DDL supports is *constraint-based* representation. Models can express interval temporal constraints among state variable values and relations between specific parameters of state variable values, i.e., arguments of the predicates that represent these values. Ideally, constraints should always be expressed *declaratively*, so as not to force a specific model of problem solving (e.g., temporal projection, backward chaining). In practice, it is often difficult to develop a model without already having in mind a problem solving method to use. HSTS-DDL allows the expression of constraints as functions returning specific predicate argument values once values of other temporally related predicates are known. Such functional attachments introduce a definite order in which a planner/scheduler should try to satisfy the model constraints.

To give an example of a complete pattern of temporal constraints, we list below an HSTS-DDL fragment from our current Cassini model (see section 5.2. for a description of the Cassini application). This compatibility rule describes a constraint on the state variable (SSR operating_status), which corresponds to the operating state of Cassini's internal tape recorder. The constraint expresses the conditions under which the tape recorder's state can assume a value (read_out) corresponding to reading out data from the detector (for example, the camera) of some instrument. Important parameters of this read-out state of the tape-recorder are the length of the current exposure, the type of the detector being recorded, and the amount of data already stored on the tape recorder.

The constraint is broken up into three conditions. The argument of the second constraint involving stored_capacity, which corresponds to the final amount of data stored on the tape recorder after this record event, is computed by executing the given Lisp code fragment once the values of the variables are known. This can happen only if temporal projection is used to determine the temporal evolution of the state variable (SSR operating_status), because the Lisp addition function is not invertible.

```
(Define_Compatibility
  ;; Constrained token
  ((SSR operating_status)
   (read_out (?detector ?exp_length ?existing_data)))
  :compatibility_spec
  ;; Constraints on the token
  (AND
   ;; (1) Predecessor value on state variable
   (met_by
    ((SSR operating_status)
     (stored_capacity (?existing_data))))
   ;; (2) Successor value on state variable
   (meets
    ((SSR operating_status)
     (stored_capacity (Lisp (+ ?existing_data
                      (compute_data_production
                         ?detector ?exp_length))))))
   ;; (3) Parallel condition on state of another variable
   (contained_by [0 _plus_infinity_] [0 0]
    ((?detector data_status)
     (data_stored (?detector))))))
```

The functional attachment mechanism is aimed at achieving a practical balance between versatility of the model and support to prototype development. A quickly developed but more restricted system can give useful empirical feedback. Such feedback can be incorporated into a more general and flexible model at a later stage.

4.2. Integrating with the User's Environment

External tools and knowledge sources are routinely used by human schedulers when building spacecraft science plans and sequences. If such tools and knowledge are not integrated within an AI planner or scheduler, it is unlikely that the tool will be used in practice. Two important integration issues concern external geometric knowledge and user interfaces. This section discusses these in more detail.

As shown in Section 4.1., HSTS allows the user to make reference to functions not directly interpretable in the HSTS-DDL. This is the natural place where *external software* can be integrated within the model. Our scheduler can make use of extensive external subroutines for different kinds of reasoning. A prime example is JPL's NAIF toolkit, an extensive FORTRAN library of geometric subroutines used for planetary calculations. The routines enable scientists to determine, for example, how long light will take to travel from a source to an instrument, or whether a given planet will be visible to the spacecraft at a specific point in its mission. To facilitate the use of these routines, we use AMPHION (Lowry *et al.* 1994), an automatic programming tool which compiles graphical, high-level, declarative specifications into a sequence of NAIF subroutine calls.

A second integration issue involves graphical user interfaces. We are exploring ways in which HSTS could work with a variety of available user interfaces, rather than requiring a new interface to be developed for each scheduler or mission. This leads to the need to consider the functionalities common to the planning or scheduling task itself, and has prompted us to work toward an *interlingua* through which schedulers and generic scheduling interfaces can communicate without knowing the details of the other program.

HSTS currently uses DTS, the customizable scheduling interface being developed by HRI (Hansson & Mayer 1994). By "interlingua," we mean that HSTS and DTS communicate at the abstract level of scheduling data, not at the detailed level of how the data should be displayed to, or manipulated by the user. When HSTS updates the start-time of an exposure, it sends DTS a simple message to that effect, and DTS decides how this should affect the display.

The overall goal of this integration effort is to provide a scheduling tool which is embedded in the user's environment and which makes use of the knowledge already there, provides a common look and feel across tasks, and is customizable to the needs of the user. This type of integration will be necessary for any planning or scheduling tool to be used in the real world.

5. Applications

HSTS is currently being used to support a number of space scheduling applications. The main applications at present are the Extreme Ultraviolet Explorer (EUVE) and Cassini. We are also gearing up to use HSTS to support NASA's New Millennium spacecraft project, which will apply advanced spacecraft technology to enable new mission scenarios with radically decreased operations costs.

5.1. EUVE

The Extreme Ultraviolet Explorer (EUVE) (Bowyer 1990; 1994) is an earth-orbiting telescope which makes observations in the wavelength range of 70 to 760 angstroms. The EUVE is operated by the NASA Goddard Space Flight Center and the Center for EUV Astrophysics (CEA) at the University of California, Berkeley. In the upcoming EUVE Extended Mission, CEA will take over all possible operations responsibility for the spacecraft, without corresponding budget increases. Because automation is the only means toward this ambitious goal, EUVE has become a testbed for automated low-cost operations in future NASA missions.

Automated science scheduling is one important step in automating EUVE operations. We have installed a manual science scheduler at CEA, which assists their experts in generating science plans (observing schedules at the level of pointings, rolls, instrument configuration and exposure times). By the end of 1995, the science scheduling process will be fully automated.

The constraints in this problem are determined by the positions of observational targets, the position and attitude of the spacecraft platform, and the positions of obstacles such as planets, the sun and atmospheric anomalies. A number of CEA preprocessors check whether these constraints are violated by combinations of pointings, rolls and desired exposure times. More complex observations, including dithered targets and moving targets, require quite sophisticated constraint-checking. For moving targets, the constraint-checkers can require over a CPU day to verify that an exposure will not violate spacecraft constraints. These detailed checks by the science planner are vital because if a constraint-violation were to be detected by the spacecraft instead, it would jeopardize spacecraft health *and* cause the spacecraft to enter a "safe-hold" position, causing lost science time and increased operations costs.

Many scheduling systems take an extremely narrow view of the scheduling process. They schedule a fixed set of observations, under prespecified constraints. But the EUVE problem is very dynamic: it involves variable slew times, multiple exposures, pointings that match an ephemeris, etc. The problem also requires quite idiosyncratic constraint-checking by a number of external programs, as well as integration with a number of proposal databases, star catalogs, etc. By one recent count, CEA uses 14 separate programs during its scheduling process. Inefficiencies result because these 14 programs are run in a manually-controlled "pipeline." By integrating these programs within HSTS, we are not only automating the process, but in fact, reducing the computational requirements, as these auxiliary programs are invoked *as needed* to provide information or check constraints.
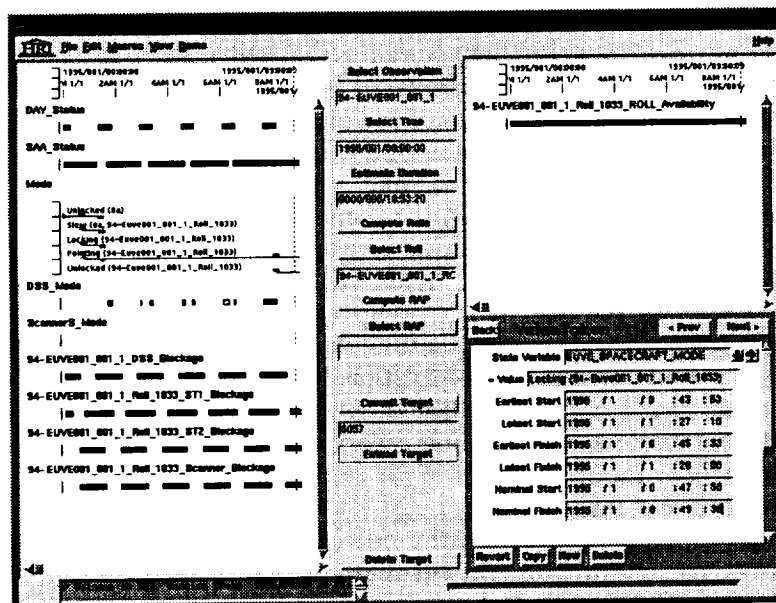

Figure 2.   The EUVE scheduler.

Finally, using the DTS interface, we are attempting to capture the scheduling process that has been developed at CEA. Figure 2 shows the interface developed for EUVE. This interface was customized for use by CEA within one man-month of programmer effort.

The screen is dominated by two DTS "dynagraphs" (dynamic graphs) displaying timelines. The left dynagraph shows details of the current schedule. Reading from the top of the screen, the interface displays day/night status, South Atlantic Anomaly (SAA) status, spacecraft mode (e.g., pointing, slewing, etc.), exposures, and then various star-tracker and instrument blockages. The right dynagraph displays information on whether the possible spacecraft roll angles violate constraints. The lower-right portion of the screen is a form displaying details on objects selected by

the user using the mouse: for example, when an exposure is selected, this form displays start-times and end-times to the second.

In the center of the screen is a vertical column of buttons, which embody the typical scheduling process. From the top, the buttons permit the user to (1) select a target to schedule, (2) choose an earliest start-time or a gap in the schedule, (3) invoke an external program to estimate elapsed time to perform the observation, (4) compute constraint-checks on all possible roll angles, (5) choose a specific roll angle, (6-7) compute and select possible targets to select in another spacecraft instrument, and finally, (8) commit the observation of this target to the schedule. This "fill-in-the-blanks" representation of the science scheduling process has greatly simplified the interface.

5.2. Cassini

Cassini is a large spacecraft with destination Saturn and Titan. The project is undergoing joint development by NASA, the European Space Agency and the Italian Space Agency. The U.S. portion of the mission is managed for NASA by the Jet Propulsion Laboratory. The date of launch is October 1997; after 7 years of cruise, Cassini will arrive at Saturn in June 2004 and will orbit the planet some five dozen times over a 4 year mission. During this time, Cassini will perform experiments toward several science objectives including the exploration of Saturn's and Titan's atmosphere and the study of Saturn rings and moons.

While the number of instruments carried by the Cassini spacecraft (12) is comparable to that of Voyager, the mission has much higher complexity. Cassini is expected to make at least 50,000 observations during its 4 year mission. While the amount of science to be performed will increase by almost two orders of magnitude, budget limitations are likely to reduce the number of support schedulers by an order of magnitude, leaving scientists responsible for most of their own sequencing.

In addition to higher complexity due to increased science and reduced personnel, Cassini is also more complex than previous missions because of the ongoing re-planning necessitated by Cassini's continuous observations. Experience has shown that periods of intense observation give rise to the highest workload and stress levels for scientists and schedulers. This is because new information becomes available which was not present at the time the initial schedule was constructed. In such cases, re-planning may be necessary either to avert a potential failure of existing goals (as when a spacecraft drifts too much from its original course) or to take advantage of unanticipated science opportunities (such as the discovery of rings around Neptune, which happened after the main scheduling was done for the Voyager Neptune encounter). Whereas previous missions were characterized by short periods of intense observations and, hence, limited periods of re-planning (when the spacecraft was near a planet), intense observations for Cassini will be the characteristic mode of operation during its four-year mission. Throughout all of this period, scientists will be subject to continuous stress caused by the need for re-planning.

In summary, while many missions had large staffs, smaller science objectives, and shorter periods of high-pressure re-planning, the Cassini mission will have a greatly reduced staff, greatly increased science objectives, and a four-year sustained period of high-pressure re-planning. While scheduling each Voyager encounter could be seen as four years of relative calm followed by three weeks of panic, without advances in mission operations software, scheduling Cassini is likely to be four years of total panic.

We are currently collaborating with the Cassini mission and more specifically we are supporting the activities of the Cassini rings and dust science working group. Although the construction of the spacecraft has not been completed and several details of its behavior are still unknown, scientists are already evaluating orbital geometries with respect to their potential scientific output. An essential tool of this evaluation is the generation of representative schedules of experiments and observations. To this end, we have used HSTS to build a scheduler for the Cassini mission (Figure 3). The scheduler has been used to generate a representative schedule of observations of the Saturn rings during the 14th orbit of Cassini around Saturn. The current HSTS-DDL model of the Cassini domain keeps track of the visibility of the Deep Space Network (DSN) stations (Goldstone, Canberra and Madrid) and the occurrence of other significant mission events, such as star occultations and flybys of Titan

and other satellites. As with EUVE, we model the observing state of all instruments and the slewing and pointing state of the spacecraft. Additionally we model the storage of scientific data in the on-board solid state recorders and the down-link of data to Earth through DSN stations.
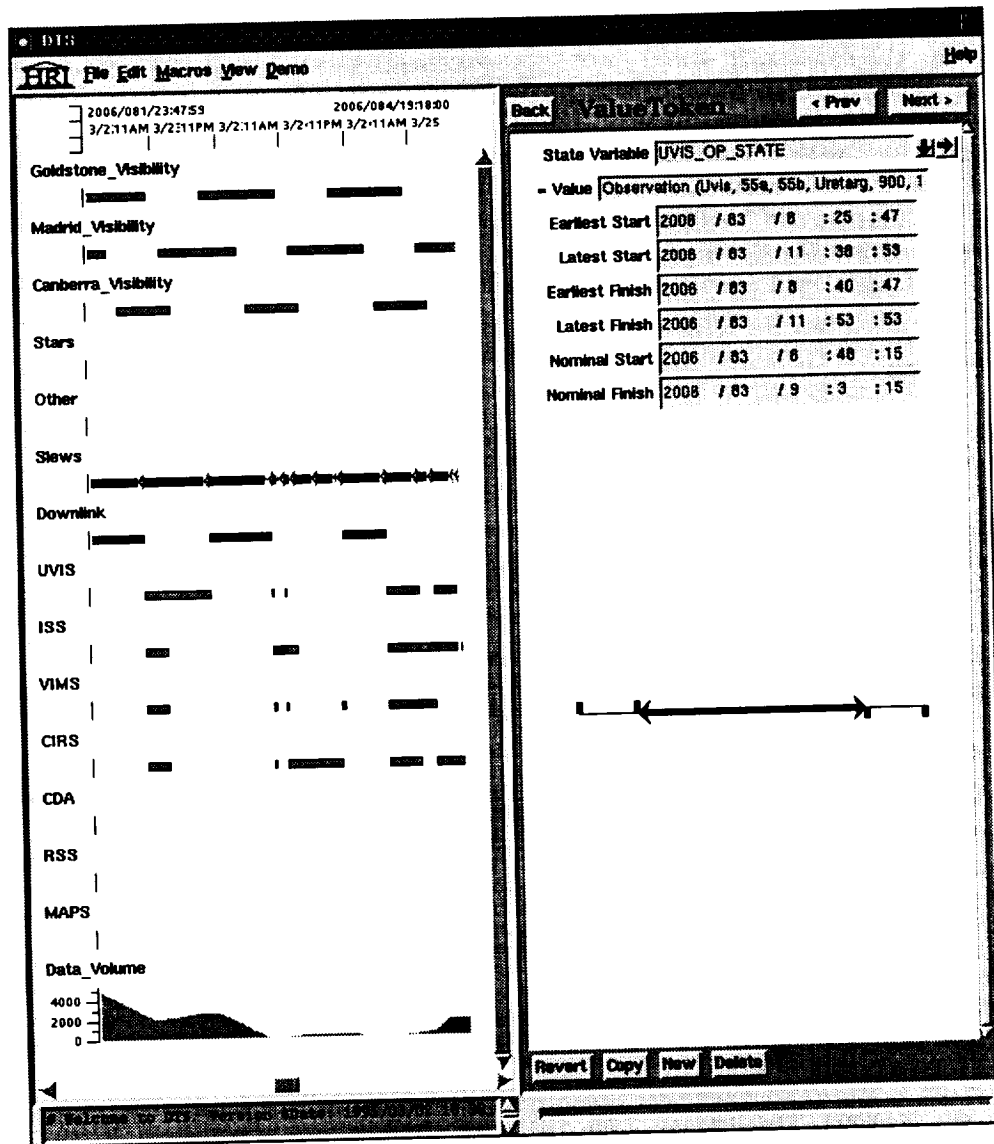

Figure 3.   The Cassini scheduler.

5.3. New Millennium

NASA has recently announced the New Millennium program. The program will take the form of a series of aggressive technology-demonstration missions. The goal of the program is to open a new mode of space exploration. This new mode will consist of "faster, smaller, cheaper" spacecraft (as described by NASA Administrator Dan Goldin), with the goal of eventually establishing a "virtual presence" in space.

    With an emphasis on low-cost operations to support a fleet of spacecraft, New Millennium is encouraging the application of advanced automation techniques to support ground operations and on-board autonomy.

In terms of science scheduling, one proposal is to move much of the functionality of scheduling and sequencing on-board the spacecraft. In this model, the scientists would communicate high-level science goals directly to the spacecraft. The spacecraft would then perform its own science planning and scheduling, translate those schedules into sequences, verify that they will not damage the spacecraft, and ultimately execute them without routine human intervention. In the case of error recovery, the spacecraft would have to understand the impact of the error on its previously planned sequence and then reschedule in light of the new information.

We are currently investigating ways to extend HSTS to meet the challenges of increased autonomy and reliability within a preliminary demonstration of an autonomous mission concept. Any automated solution will require strong modeling capabilities, powerful algorithms for constraint processing and search, and facilities for integration with a multitude of external software routines. We are confident that the development of these components of HSTS in support of existing missions like EUVE and Cassini will enable increasingly automated scheduling for New Millennium.

6. Conclusion

This paper has described our experience to date in developing a planning and scheduling framework, HSTS, to support scientific space missions. Space mission scheduling is a good example of a practical problem with important repercussions on the cost and, therefore, the feasibility of future space exploration.

The automation of space mission scheduling poses several challenges to computer science. Key among these are supporting the entire process of scheduling rather than just a portion of it, and developing powerful modeling languages that can deal with large models and diverse sources of information. We have addressed these challenges in applying HSTS to EUVE and Cassini mission scheduling.

Acknowledgements

References

AAAI. 1994. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Seattle, WA: AAAI/MIT Press.

Atkinson, D., ed. 1994. *Proceedings of the Third International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*.

Bowyer, S. 1990. The extreme ultraviolet explorer mission. In Kondo, T., ed., *Observatories in Earth Orbit and Beyond*. Kluwer Academic Publishers. 153–169.

Bowyer, S. 1994. Extreme ultraviolet astronomy. *Scientific American* 271(2):32 ff.

Bresina, J. L. 1994. Telescope loading: A problem reduction approach. In Atkinson (1994).

Drummond, M.; Bresina, J.; and Swanson, K. 1994. Just-in-case scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (1994), 1098–1104.

Fikes, R.; Hart, P.; and Nilsson, N. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Fox, M., and Zweben, M., eds. 1994. *Intelligent Scheduling*. Morgan Kaufmann.

Hanks, S., and McDermott, D. 1994. Modeling a dynamic and uncertain world i: Symbolic and probabilistic reasoning about change. *Artificial Intelligence* 66(1):1–55.

Hansson, O., and Mayer, A. 1994. DTS: Building custom, intelligent schedulers. In Atkinson (1994).

Johnston, M. D., and Miller, G. E. 1994. SPIKE: Intelligent scheduling for hubble space telescope observation. In Fox and Zweben (1994).

Lansky, A. 1988. Localized event-based reasoning for multiagent domains. *Computational Intelligence Journal* 4(4). Special Issue on Planning.

Lansky, A. 1994. Localized planning with diverse plan construction methods. Technical Report FIA-94-05, NASA Ames Research Center.

Lowry, M.; Philpot, A.; Pressburger, T.; and Underwood, I. 1994. Amphion: Automatic programming for scientific subroutine libraries. In *International Symposium on Methodologies for Intelligent Systems*.

Minton, S.; Johnston, M. D.; Philips, A. B.; and Laird, P. 1992. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58:161–205.

Muscettola, N. 1994a. HSTS: Integrating planning and scheduling. In Fox and Zweben (1994).

Muscettola, N. 1994b. On the utility of bottleneck reasoning for scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence* (1994), 1105–1110.

Smith, S. F. 1994. OPIS: A methodology and architecture for reactive scheduling. In Fox and Zweben (1994).

Vere, S. 1983. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5:246–267.

Zweben, M.; Davis, E.; Daun, B.; and Deale, M. 1993. Scheduling and rescheduling with iterative repair. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6):1588–1596.