NASA-CR-202425

Investigations into Generalization of Constraint-Based Scheduling Theories with Applications to Space Telescope Observation Scheduling

Final Report for NASA Contract # NCC 2-531

Submitted to:

NASA Ames Research Center
University Grants Office
Mail Stop JAC:241-1
Moffett Field, CA 94035-1000
Attn: Sonie Lau

Prepared by:

Nicola Muscettola and Stephen F. Smith
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

September, 1996

Executive Summary

This final report summarizes research performed under NASA contract #NCC 2-531 toward generalization of constraint-based scheduling theories and techniques for application to space telescope observation scheduling problems. Space telescope observation scheduling, like many complex scheduling problems, requires attendance to several classes of interacting constraints. At one level, it is necessary to reconcile the requirements of a large and conflicting set of user requests with the overall objective of optimizing resource utilization and maximizing scientific return. However, allocation decisions must also satisfy complex physical constraints that introduce interactions between the behavior of the various components of the system and its environment. These constraints actually dictate the circumstances under which specific user requests can be achieved and introduce the need for planning, synchronizing, and allocating resources to auxiliary enabling system activities. In short, these problems require integration of scheduling and plan generation capabilities.

Our work into theories and techniques for solution of this class of problems has led to development of HSTS (Heuristic Scheduling Testbed System), a software system that provides a unified framework for integrated planning and scheduling. Within HSTS, planning and scheduling are treated as two complementary aspects of the more general process of constructing a feasible set of behaviors of a target system. We have validated the HSTS approach by applying it to the problem of generating observation schedules for the Hubble Space Telescope.

This report summarizes the HSTS framework and its application to the Hubble Space Telescope domain. First, the principal components of the HSTS software architecture are described, indicating (1) how the structure and dynamics of a system is modeled in HSTS, (2) how schedules are represented at multiple levels of abstraction, and (3) the problem solving machinery that is provided. Next, the specific scheduler developed within this software architecture for detailed management of Hubble Space Telescope operations is presented. Finally, experimental performance results are given that confirm the utility and practicality of the approach.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

Automated robotic systems offer new opportunities in many diverse areas, from flexible manufacturing to space exploration. However, the extent to which their potentiality can be actually exploited depends on the ability to efficiently manage their complex operations. The specific domain of focus in the research described in this paper - coordinating the use of the Hubble Space Telescope (HST) - is representative of such system management problems. Efficient operation of such automated systems requires attendance to several classes of interacting constraints. At one level, it is necessary to reconcile the requirements of a large and conflicting set of user requests with the overall objective of optimizing resource utilization. However, allocation decisions must also satisfy complex physical constraints that introduce interactions between the behavior of the various components of the system and its environment. These constraints actually dictate the circumstances under which specific user requests can be achieved and introduce the need for planning, synchronizing, and allocating resources to auxiliary enabling system activities.

In this paper, we describe HSTS (Heuristic Scheduling Testbed System), a software system architecture for integrated planning and scheduling that has been applied to the problem of constructing short-term executable observation schedules for the HST. HSTS is based on a view of resource allocation (scheduling) and goal expansion (planning) as complementary aspects of a more general process of constructing a behavior (or set of behaviors) of a dynamical system that is consistent with stated goals and constraints [1]. The HSTS software architecture provides three principal capabilities:

1. a domain description language for modeling the structure and dynamics of the physical system at multiple levels of abstraction
2. a temporal data base for representing possible evolutions of the state of the system over time, and
3. a scheduling/planning framework for integrating decision-making at multiple levels of abstraction

Using the HSTS system, an incremental approach to the HST problem has been adopted, through development and analysis of increasingly more complex models of the telescope operating environment. The current HST scheduler operates with a model that captures most of the complexity of the actual operating environment, and efficiently generates detailed sequences of system reconfiguration activities that reflect global allocation objectives. At the same time, the HSTS modeling and problem solving framework is quite general, and is applicable in any planning domain requiring efficient allocation of resources in the presence of complex physical constraints. In comparison to traditional software approaches, the HSTS framework advocates explicit separation of data and control, providing flexibility with respect to changes in system dynamics and the external environment.

The HST is a sophisticated $1.4 billion observatory that was placed into low earth orbit in early 1990. With its complement of six viewing instruments, HST allows astronomers to observe and analyze celestial objects at a distance 7 to 10 times further than is currently possible from existing ground-based observatories. Potential astronomer demands for telescope viewing time are virtually unlimited, and maximization of scientific return over the telescope's expected operational lifetime of 15 years is of fundamental importance. Contention for viewing time among prospective users is high, and efficient management of telescope operations to maximize scientific usage is thus a critical operational concern.

Current support for HST operations is provided by the Science Operations Ground System (SOGS). At the heart of SOGS is a FORTRAN-based software system called SPSS, originally envisioned as a tool which would take astronomer viewing programs as input (accepted from competing proposals on an annual basis) and produce executable spacecraft instructions as output. SPSS has had a somewhat checkered past [2], due in part to the complexity of the scheduling problem and the constraints that must be taken into account, and in part to the difficulty of developing a solution via traditional specification-based software engineering practices and conventional programming languages. Evolving specifications of various telescope capabilities and operational constraints, for example, have necessitated several rewrites to major portions of the code over its 9 year development cycle. As a basis for supporting HST operations, SPSS has exhibited several shortcomings:

- inflexibility - SPSS implements a rigid approach to solving the problem. The treatment of different constraints is inextricably bound to the underlying scheduling algorithm. Unanticipated constraints cannot be easily accommodated, and many sources of scheduling flexibility admitted by the current HST proposal specifications [3] simply cannot be exploited. This, in turn, places limits the system's ability to maximize utilization of the telescope.

- incompleteness - Many detailed physical constraints are not adequately modeled in SPSS, requiring a significant amount of manual post-processing (i.e. constraint checking) of the schedules that are produced before they can be executed.

- scope - Schedules are developed non-hierarchically with respect to all embedded operational constraints by SPSS, making the generation of schedules over horizons longer than one or two weeks computationally infeasible.

To confront computational problems, the Space Telescope Science Institute - the agency responsible for overall management of telescope operations - has developed a separate, Artificial Intelligence (AI) based tool for long term scheduling called SPIKE [4]. SPIKE is used to partition the requests of approved observation programs into weekly time buckets and currently serves as a front end to SPSS short-term scheduling. Detailed short-term schedules are currently generated through the efforts of a sizable group of operations astronomers, who interactively guide the placement of observations on the time line by SPSS.

The research described in this paper aims at providing a more effective solution to the HST short-term scheduling/planning problem through integration and extension of recent research in AI-based planning and scheduling. In contrast to the SPSS solution, the work reported here has emphasized the development of a framework for flexibly representing and manipulating complex constraints, as well as the development of heuristic strategies for effectively balancing conflicting scheduling constraints and objectives. It is important to note that the HST problem is, in fact, representative of a larger class of space mission planning problems. Many of these other problems (e.g. coordination of space station activities) are considerably more complex, and inflexible solution approaches that rely heavily on human intervention and participation will not be possible. Thus, the importance of better approaches to detailed, short-term scheduling transcends the HST domain.

As indicated above, the characteristics of the short-term HST scheduling problem require an integration of what have historically been distinguished as "scheduling" and "planning" techniques, as each offers specific strengths with respect to the required overall process. Research in scheduling has classically focused on optimal solutions to idealized resource allocation problems [5] and local heuristic dispatching rules designed to attend to specific allocation objectives [6]. More recent research in constraint-based scheduling [7-9] has

emphasized the problem of efficiently allocating resources to competing activities over time in the presence of conflicting objectives and preferences, and has produced heuristic techniques that exploit the structure of the problem constraints (in particular, bottleneck analysis) to opportunistically focus solution development toward an acceptable global compromise. The power of these techniques vis a vis classical dispatch-based approaches has been demonstrated in large-scale manufacturing scheduling contexts [10]. At the same time, the ability to exploit such problem structure relies on specific representational assumptions held in common with classical manufacturing scheduling research [5]. In particular, it is assumed that physical constraints are "pre-compilable" so that the complete set of activities requiring resources, as well as their ordering relationships and durations, are known in advance. This leaves resource availability as the only aspect of state that must be attended to over time, and permits a model of resource availability wherein a resource is considered unavailable during any interval that it is allocated to an activity and otherwise available. These representational assumptions are insufficient in domains like HST scheduling where the ability to execute a given observing request is a complex function of the state of the underlying physical system and variably implies different networks of supporting activities and resource requirements. In such domains, techniques based on these assumptions can at best provide guidance in focusing the development of an executable schedule.

AI-based research in planning, alternatively, has focused on the problem of "compiling" activity networks that bring about desired goal states from more basic representations of the effects of actions in the world. However, as with scheduling research, the techniques that have emerged do not fully address the requirements of the class of problems described above. The appropriateness of classical STRIPS-style representational assumptions [11,12] is limited, given the absence of primitives for expressing temporal constraints and the obvious need to deal explicitly with time (in both absolute and relative senses). More recently developed representational frameworks [13-16] do provide these capabilities. However, with few exceptions (e.g. [15]), these frameworks have not attempted to exploit the inherent structure of the underlying physical system of interest. Given the complexity of systems like HST, the ability to work with decomposable models of system behavior is fundamental to managing the combinatorics of search. More generally, current planning representations and frameworks do not provide a convenient basis for reasoning globally about efficient resource allocation. Interactions in resource requirements emerge only as the represented physical constraints are applied to achieve planning goals. Allocation conflicts can be avoided, but there is no leverage to anticipate resource contention, compromise among conflicting objectives and dynamically organize planning on this basis.

The remainder of the paper describes the HSTS software system architecture for integrated planning and scheduling and its application to the HST short-term scheduling problem. First, a description of the HST operating environment is given. Then, the description language is presented and its use is illustrated by detailing the current model of the HST operating environment. An overview of the integrated HSTS scheduling and planning framework is presented next, followed by a description of the heuristics implemented in the current HST scheduler. Finally, performance results obtained on a realistically-sized observation scheduling problem are given, and the implications of our solution are discussed.

2. THE HST DOMAIN

Before we introduce the HSTS approach to modeling complex systems, let us give a brief overview of the HST operating environment.

Considered as a whole, HST is a very sophisticated "sensor"; it gathers light from celestial objects, named targets, and communicates scientific data back to Earth through one of two TDRSS communication satellites (Figure 2-1). Since the telescope is in low earth orbit, both the targets and the satellites are periodically occulted by the Earth.

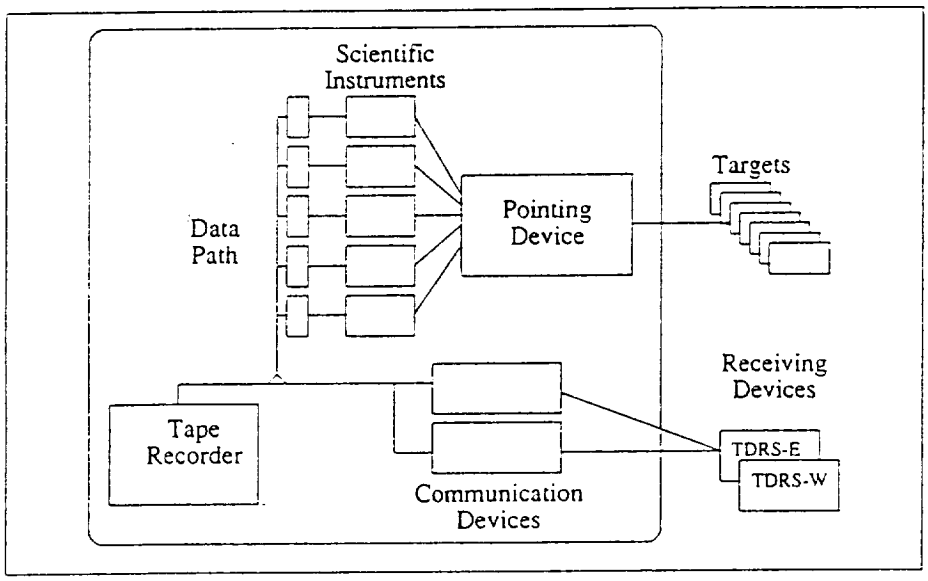

Figure 2-1: The Hubble Space Telescope Domain

The telescope itself consists of several components; their behavior must be coordinated over time due to the limitation of available electric power, the need to maintain acceptable temperature profiles on the telescope structure, etc. The pointing device represents the HST subsystem responsible for orienting it in the direction of a target and locking the target at the center of the field of view of a designated scientific instrument. HST has 6 different scientific instruments. Limitations on available electric power prevent all of them from being switched on simultaneously. Data can be read from the instruments and directly communicated to Earth through one of two links operating at different communication rates (1Mb/sec and 4kb/sec); data can also be temporarily stored on an on-board tape recorder and communicated to Earth at a later time.

Astronomers formulate observation programs according to a fairly sophisticated specification language [10]. The basic structure of each program is a partial ordering of observations, each specifying the collection of light from a celestial object with one of the telescope's six scientific instruments. A diverse set of temporal constraints can be imposed on the observations in a

program, including precedences, windows of opportunity for groups of observations, minimum and maximum temporal separations, and coordinated parallel observations with different viewing instruments.

Solving the HST observation scheduling problem requires the generation of the sequence of commands needed both to carry out observations (e.g., "take an exposure", "communicate to Earth data from instrument X") and to reconfigure the telescope to enable observation execution (e.g., "point telescope to target Y", "switch on instrument Z").

3. MODELING COMPLEX SYSTEM DYNAMICS IN HSTS

In this section we introduce the primitives of the HSTS Domain Description Language (HSTS-DDL). The HSTS-DDL supports modeling of a system at multiple levels of abstraction as a collection of interacting components.

An HSTS model is subdivided into a set of **system components**, each of which has an associated set of **properties**. At any instant of time, each property of the system can have one and only one associated **value**. Some properties are **static**, i.e., their value does not change over time. Others are **dynamic**, i.e., usually their values change over time; in the following we will also refer to these as **state variables**.

The HSTS Domain Description Language requires explicit declaration of the set of possible values that can be assumed by each dynamic property. In general, a value has the form $R(x_1, x_2, ..., x_n)$, where $<x_1, x_2, ..., x_n>$ represents a tuple in the relation R.

A **behavior** of the system is an evolution over time of the values of its state variables: it is completely specified once each state variable has an associated value for each instant of time. In a behavior, each state variable changes its value a discrete number of times and a value persists for a continuous interval of time.

To fully describe a value, HSTS-DDL requires the specification of its **duration**, a constraint on its temporal length due to the value's intrinsic characteristics. A duration is a pair of temporal distances $[d, D]$, $D \geq d \geq 0$ where d and D are respectively the lower bound and the upper bound of the duration: in general both may be functions of some arguments of the value.

The complexity of the dynamics of a system stems from the interactions between different state variables. In fact, a value can be present in a behavior of the system only if well-defined patterns of values occur over time on the state variables. In HSTS-DDL these patterns are specified by associating a **compatibility specification** with each value. A compatibility specification consists of one or more sets of compatibilities organized as an AND/OR graph. A **compatibility** associated to a (constrained) value specifies how this value must be temporally related to another (constraining) value or sequence of values. More precisely, a compatibility is a 4-tuple:

<comp–class, st–var, type, temp–rel>

where *comp–class* is either the symbol VALUE or the symbol SEQUENCE, indicating whether the compatibility involves a single constraining value or a constraining sequence of contiguous values; *st–var* is the state variable of the constraining value or sequence; *type* indicates the subset of values from which the constraining value or sequence is extracted; and *temp–rel* is a temporal relation that specifies a pattern of distance constraints. For example, the temporal relation

before([*d*, *D*]) means that the end of the constraining value/sequence must precede the start of the constrained value by an interval of time δ, such that $d \leq \delta \leq D$. The relation *contains*([d_1, D_1], [d_2, D_2]), indicates that the constrained value must be contained within the constraining value/sequence; [d_1, D_1] defines the distance between the two start times and [d_2, D_2] defines the distance between the two ends. In the next section we will give several examples of compatibility specifications.

A model represented in HSTS-DDL can be subdivided into a number of **layers of abstraction**. An abstract model consists of system components and state variables that aggregate several components and state variables at more detailed levels. The relationship among the layers is established by **refinement descriptors** that map some of the values associated with an abstract layer into a network of values associated with the immediately more detailed layer. The mapping also specifies the correspondence between the start and end times of each abstract value and those of the corresponding detailed values.

4. THE HST DOMAIN MODEL

Within HSTS, we have developed a model of the HST operating environment. At present, it includes 2 of the 6 HST scientific instruments, telescope pointing, data communication to Earth and the on-board tape recorder. The components of this model are summarized below.

The environment external to the telescope consists of targets and TDRSS satellites. For each of them, a distinct state variable characterizes its visibility with respect to the space telescope.

The possible values of the pointing apparatus (state variable HST-POINTING) are: *LOCKED*(?*T*), *UNLOCKED*(?*T*), *LOCKING*(?*T*), and *SLEWING*(?*T*1, ?*T*2). Here ?*T*, ?*T*1, and ?*T*2 are variables denoting possible targets. For example, *SLEWING*(?*T*1, ?*T*2) represents the movement of the telescope from the direction pointing to ?*T*1 to that pointing to ?*T*2. Figure 4-1 shows the possible transitions among values. An edge connects two nodes if the corresponding values are related by a *before*([0,0]) compatibility. The highlighted nodes represent "stable values", i.e., values with intrinsic duration [0, +∞]; all the other nodes represent states with finite durations.

During telescope repointing, some of the values can occur only while a target is visible. More precisely, a window of visibility for target ?*T* must necessarily contain the occurrence of a *LOCKING*(?*T*) operation and of a *LOCKED*(?*T*) state. Figure 4-2 lists the complete compatibility specification for *LOCKED*(?*T*). It indicates that in every correct behavior of HST, *LOCKED* can appear on HST-POINTING either in a sequence *LOCKING*, *LOCKED*, *UNLOCKED* or in a sequence *LOCKING*, *LOCKED*, *SLEWING*, together with the constraint of co-occurrence between *LOCKED* and a target visibility window.

Of the two instruments currently modeled, the Wide Field Planetary Camera is the most complex. It consists of two different sets of CCD sensors, or detectors, the Wide Field Camera (WF) and the Planetary Camera (PC). Both share the same support module that provides them with temperature control, optical filters reconfiguration operations, etc.; we will denote the support module with WFPC. Each of the three components of the Wide Field Planetary Camera is modeled as a separate state variable. Figure 4-3 shows the value transition graph for the WFPC-STATE state variable; similar graphs describe the value transitions of WF-STATE and PC-STATE.

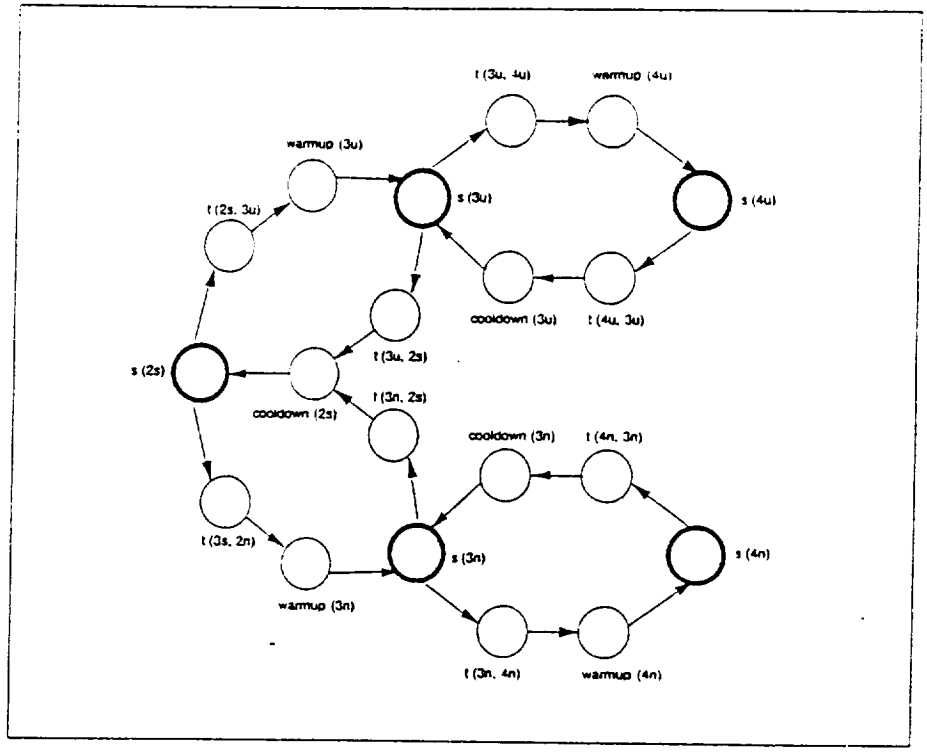

Figure 4-3: Value transition graph for WFPC-STATE

The value *s(2s)* represents the state where the Wide Field Planetary Camera is switched off, and the values *s(4u)* and *s(4n)* represent fully operational states. Both WF-STATE and PC-STATE can assume, as a possible value, *EXPOSE*(*?config*, *?target*, *?duration*), which represents a picture taking operation; here the variable *?config* can be either 4*u* or 4*n*, depending on the operational configuration required on the WFPC. Figure 4-4 expresses the compatibilities that have to be satisfied for the occurrence of *EXPOSE*(4*n*, *?target*, *?duration*) on the WF-STATE.

Further physical constraints require synchronization of the warmup and cooldown processes of each of the three instrument components. For example while the WF-STATE has value *s(3n)*, WFPC-STATE can undergo any warm-up or cool-down sequences between *s(3n)* and *s(4n)*; this is represented as a compatibility associated with the *s(3n)* of the WF-STATE (Figure 4-5).

Data are read out of a detector either by the tape recorder or by one of the two communication links. The values of TAPE-RECORDER-STATE indicate that the quantity of stored data increases at each read-out operation and that the content of the tape recorder has to be "dumped" to Earth when a given capacity threshold is exceeded. The read-out operations on the communication links and the dump operation on the tape recorder must occur during a window of visibility of one of the two TDRSS satellites.

To provide a global perspective on HST operation, the detailed model just described is augmented with an abstract model. Here the whole telescope is described as a single state variable, HST-STATE. The possible values of the abstract state variable are *OBSERVE*(*?P*, *?I*, *?S*, *?T*, ...) (where *?P* designates an observation program, *?I* designates a viewing instrument, *?S* designates the required operating state of *?I*, and *?T* designates a target), *RECONFIGURE*(*?FROM–I*, *?TO–I*, ...), which represent the reconfiguration process between

```
AND [ <VALUE, HST-POINTING,LOCKED (?target), contains ([0, +∞], [0, +∞])>,
      <VALUE, PC-STATE,S (2s), contains ([0, +∞], [0, +∞])>,
      <VALUE, WFPC-STATE, S (4n), contains ([0, +∞], [0, +∞])>]
```

Figure 4-4: Compatibilities for *EXPOSE*(4*n*, ?*T*, ?*D*)

```
< SEQUENCE,
  WFPC-STATE,
   { S (3n), T (3n, 4n), WARMUP (4n), S (4n), T (4n, 3n), COOLDOWN (3n) },
    contains ([0, +∞], [0, +∞])>
```

Figure 4-5: A sequence compatibility

two observations. and *IDLE*, which represent a stationary state of the telescope where all detectors are off and the tape recorder is empty. The value *RECONFIGURE*(*?FROM–I*, *?TO–I*, ...) provides an abstract description of an entire segment of detailed behavior: its duration can be determined by analyzing the network of values that implements it at the detailed level.

Correspondence between abstract and detailed layers is insured by a refinement descriptor that maps an abstract *OBSERVE*(*?P*, *?I*, *?S*, *?T*, ...) value into the detailed *EXPOSE* and *READ–OUT* values that actually implement it.

5. THE TEMPORAL DATA DASE

In HSTS, the construction of an executable schedule is viewed as an incremental process of constraint posting and propagation on a central data base. The HSTS Temporal Data Base (HSTS-TDB) extends the philosophy of the time map formalism developed in [17] by tightly connecting the state of the data base and the model of a system. This association provides a strong basis to support planning and enforce data base consistency.

The unit of description of temporal behavior is the **token**, a quadruple <*state–variable*, *type*, *st*, *et*,>, where *state–variable* is the identifier of one of the state variables in the system model, *type* is a subset of the state variable values, and *st* and *et* are the token's start and end times respectively. The token's meaning is that *state–variable* assumes a sequence of one or more values extracted from the set *type* during the interval of time within *st* and *et*.

Depending on the number of values that the state variable can assume within *st* and *et*, we distinguish between two different kinds of tokens:

1. *value tokens*: A value token indicates that, within *st* and *et*, *state–variable* assumes a single, constant value in *type*.

2. *constraint tokens*: A constraint token specifies that, within *st* and *et*, *state–variable* assumes a sequence of values of indefinite length (possibly empty), each belonging to *type*. During the refinement of a constraint token can be replaced by a sequence of value tokens satisfying the sequence's type constraint; therefore the constraint token is the primary mechanism for describing partially specified evolutions of state variables over time.

The representation of time in HSTS-TDB is flexible, as is the one used in CPM/PERT networks [18] and other temporal data bases [17]. The *st* and *et* of each token are considered as variables; durations and compatibilities derived from a system model and temporal requirements imposed by the user specify constraints among these variables. HSTS-TDB represents temporal information as a directed graph, with token's starts and ends associated to the nodes; a directed edge labeled with the pair $[d, D]$ from node t_i to t_j indicates the existence of a relative temporal constraint between t_i and t_j that restricts their distance to vary within the interval $[d, D]$. Absolute temporal constraints are represented as relative distances from a *reference* time constant that represents the origin of the temporal axis.

The tight connection between a network of tokens in HSTS-TDB and the corresponding system model expressed in HSTS-DDL implies that a value can appear in a physically consistent evolution of a state variable only if it satisfies the constraints imposed by the physics of the system; in order to enforce and test consistency, each value token that has been constrained to

assume exactly one value in *type* is associated with an instance of the compatibility specification graph associated with the value in the model. As we will see in the following, the planning process consists essentially of constructively demonstrating the existence of a set of behaviors that satisfies the requirements of the compatibility specifications for the tokens in the database.

HSTS-TDB also supports problem solving at multiple levels of abstractions. This is obtained by subdividing a token network into a number of communicating layers, each corresponding to a level of abstraction in the system model. Each value token whose type has a refinement specification in the system model is associated with an instance of its specification. As we will see in the following, consistency among layers of abstraction is established by connecting tokens in order to satisfy the requirements of refinement specifications.

In each layer, tokens are organized in two networks: a *goal network* and a *behavior network*. The *goal network* contains tokens and temporal constraints that describe the problem's requirements; each value token is an elementary goal and goals can be related by the same kind of temporal relations used to specify compatibilities. For example, in an HST observation scheduling program each request for an observation (e.g., observe target 3C267 with the WF in configuration 4n) is translated into a value token; relative temporal constraints among observations (e.g., take two pictures of 3C267 separated by at least one day) are implemented by temporal relations among tokens while absolute temporal constraints (e.g., take a picture within the first 15 days of October 1991) are implemented by temporal relations between the *reference* and goal tokens.

Planning at a given level of abstraction consists of repeatedly selecting goal tokens and building system behaviors that achieve them. The construction of system behaviors proceeds in a separate token network, the *behavior network*. For each state variable, a behavior network contains a linear sequence of tokens that completely covers the entire scheduling horizon. The *st* of the first token in the sequence and the *et* of the last are constrained to occur respectively at the beginning and at the end of the scheduling horizon; moreover, the *et* of a token is identical to the *st* of the following token in the sequence. If a segment of state variable behavior is only partially specified, it will be covered by one or more constraint tokens.

During the planning process, each layer of the data base is repeatedly refined through the application of three basic data base modification operations:

1. *commitment on the achievement of a goal.* This results in the insertion of a value token belonging to the goal network into the behavior network. There are two ways in which this can be accomplished.

   The first consists in merging the goal token with a matching value token in the behavior network. For example, suppose that two different observation programs require observations with matching characteristics and the behavior network already contains a plan to achieve one of them; then the achievement of the second observation can be assured by simply creating a link between the new requirement in the goal network and the observation token already existing in the behavior network.

   The second way is to insert the goal token into a compatible constraint token, i.e., one whose type has a non empty intersection with the type of the goal token. Figure 5-1 illustrates in more detail the insertion process. The partial specification provided by the initial constraint token implicitly allows a certain set of state variable trajectories; after the insertion of the goal the set of legal trajectories is

restricted to those that assume the goal value over the specified time interval.

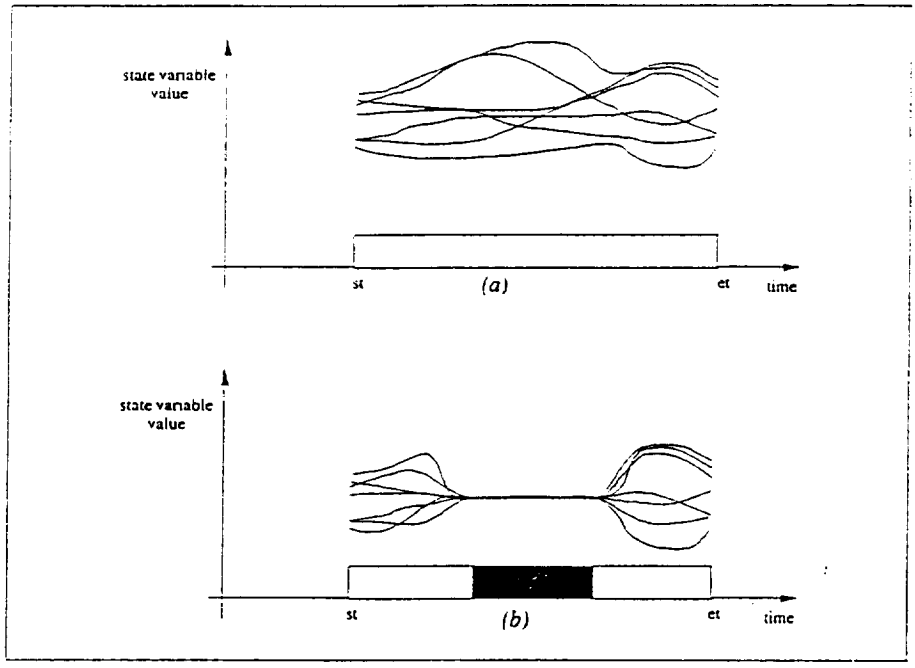

Figure 5-1: Insertion of a goal token into a constraint token: (a) before; (b) after

After the commitment, the two networks share the value token; therefore any constraint subsequently imposed on one network will propagate to the other (e.g., the expansion of an auxiliary task in the behavior network will affect how to achieve the remaining goals).

2. *value compatibility implementation.* This operation implements the requirements specified by one of the open value compatibilities of a token in the behavior network. The first step consists in the identification of a value token that satisfies the type requirement in the compatibility; the token is either selected from the existing ones or created by refining a constraint token. The second step consists in connecting the two tokens by implementing the specified temporal relation. A compatibility can be consistently implemented if the behavior contains tokens that satisfy both the compatibility's type and temporal relation. Type consistency requires the type of a token to have a nonempty intersection with the type of the compatibility. Temporal consistency requires that the network of temporal constraints not contain cycles of distance links with total length necessarily different from 0. Figure 5-2 summarizes the process of implementing a *contains* compatibility in an HST example; the compatibility specifies that while the WF is taking a picture of target 3C267, the telescope must be pointing and locked on the target.

3. *sequence compatibility implementation.* This operation is analogous to the previous one. In this case the temporal relation connects the constrained value token to a sequence of value and constraint tokens whose type matches the compatibility's type constraint. The type of each constraint token in the constraining sequence

must now also satisfy the compatibility type constraint.

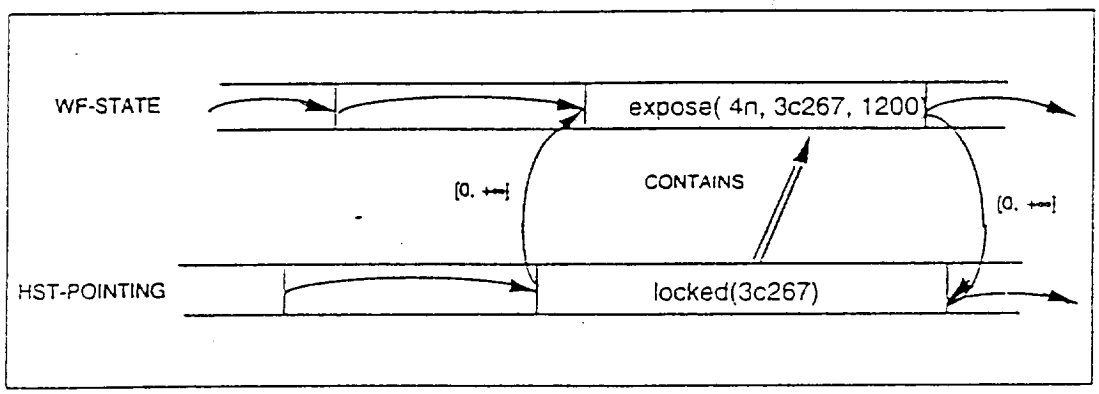

Figure 5-2: Implementation of a *contains* compatibility

The three operations mentioned above provide the primitive mechanisms to implement integrated planning and scheduling problem solvers. In fact goal commitment is analogous to a resource allocation step in a classical scheduling algorithm, while compatibility implementation corresponds to precondition and postcondition expansion and implementation found in classical Artificial Intelligence planning. In the next section we will see how these primitives are used in the HSTS integrated planning and scheduling methodology.

6. THE PLANNING PROCESS

Multiple levels of abstraction can provide significant leverage in managing the combinatorics of planning and scheduling for complex systems. Abstract views provide a basis for globally focusing the detailed planning effort; on the other hand, detailed views provide a basis for sharpening abstract predictions. The HSTS planning and scheduling methodology supports flexible integration of planning in different layers of the temporal data base.

The objective of planning/scheduling at a given layer of abstraction is to transfer goal tokens from the goal network into the behavior network and generate a consistent system behavior that achieves these goals. In HSTS this process of commitment and behavior generation is carried out incrementally by repeated applications of the planning procedure depicted in Figure 6-1.

All 4 steps in the skeletal procedure of Figure 6-1 typically require choices among alternatives. For example, a compatibility can be implemented in several ways, since in general we can select

```
PlanningProcedure (GoalNet, BehaviorNet)

    (1) select some goals from GoalNet;
    (2) insert selected goals in BehaviorNet;

    {repeat

        (3) select an open compatibility
            in BehaviorNet;
        (4) implement selected compatibility

    until no more open compatibilities }
```

Figure 6-1: The planning procedure

the constraining token or sequence of tokens in different positions within the evolution of a state variable. When different choices are possible, they are separately explored through the application of a heuristic search procedure. However, the pruning and selection criteria used by the search are not provided by HSTS, since they usually depend on the characteristics of both the application domain and the problem to be solved. The procedure constitutes only a skeleton that needs to be augmented with additional heuristic knowledge when implementing a planner/scheduler for a particular domain. In the next section we will discuss the heuristics presently used by the HST observation scheduler.

Each representational layer has an associated planning process. A planning process exchanges information with its two adjacent layers (or to the external world if a layer does not exist). Let us consider two adjacent layers i and $i+1$, where i is less abstract than $i+1$. The communication from $i+1$ to i involves the request to solve a problem, i.e., a refined goal network and preferences on how the goals should be achieved (e.g., "achieve all the goals as soon as possible"). Process i communicates back to level $i+1$ more precise information resulting from detailed problem solving, in particular additional temporal constraints on the abstract goals.

At its core, a planning process repeatedly calls the planning procedure described before, using heuristics that are most suitable to the characteristics of the system model. When a procedure returns, the process has to decide whether to continue planning at this level or to communicate information to one of the adjacent layers and pass the control to the corresponding process. Processing stops when either all the goals communicated by the user have been achieved or it has been determined that no more goals can be achieved.

The HSTS planning/scheduling methodology does not impose a specific pattern of

coordination among problem solving at different levels of abstractions. Therefore, the implementation of a problem solver for a specific application domain requires specifying how the various planning processes coordinate. The next section describes the coordination that is currently used by the HST observation scheduler.

7. OBSERVATION SCHEDULING IN THE HST DOMAIN

In this section we describe the scheduler for the HST domain currently implemented in the HSTS architecture. Some performance results obtained with the program on realistically-sized observation scheduling problems will then be reported.

Given the two-level model described in an earlier section, the current HSTS observation scheduler has two planning processes acting on each layer of a two-layer temporal data base. Planning at the abstract level has responsibility for determining the sequence of observations to be executed; planning at the detailed level is responsible for developing a detailed system behavior that implements this observation sequence.

To make use of more precise information about actual telescope reconfiguration durations as scheduling proceeds, decision-making at the abstract level and at the detailed level are tightly coupled. Each time an observation is selected and inserted into the abstract behavior network, on the basis of the abstract estimates of telescope reconfiguration times, control is passed to detailed planning that expands the detailed setup tasks to achieve the new expose and communicate goals. Abstract planning currently utilizes a dispatch-based approach (see below); on each iteration, an additional observation is appended to the abstract behavior extending the schedule strictly forward in time.

The heuristics employed by each planning process reflects an overall objective of maximizing the amount of time during which the telescope actually collects scientific data. These heuristics are described in the following two subsections.

7.1. Sequencing heuristics

At the abstract level, a single state variable characterizes the behavior of the telescope and there is only one possible path through its associated value transition graph. The complexity of decision-making at this level lies entirely in goal selection.

Selection of the next goal to achieve is accomplished by a local greedy heuristic designed to minimize dead time between observations. Specifically, a one-step look-ahead search is performed where each of the eligible candidates in the goal network are hypothesized as the next observation to be executed. A goal may be ineligible due to ordering constraints with other unachieved goals or due to absolute time constraints. Among the eligible goals, the one that yields the estimated earliest start time is identified. However, selection of this goal might prevent the subsequent achievement of one or more other goals due to some temporal constraint in the network. If this is not the case, the earliest start goal becomes the final selection; otherwise, the temporally constrained goal with the highest priority is selected, and the remaining unachievable goals are removed from the goal network.

7.2. Detailed Planning Heuristics

At the detailed level, the selection of goals results from the refinement of the choice made at the abstract level. However, heuristics guide the compatibility selection and implementation decisions.

Selection of a compatibility to implement: The compatibility specification graphs of the detailed model contain OR nodes. Once these graphs are instantiated in the temporal data base, the planner must select only one of the alternative OR branches for implementation. Heuristics perform this selection. All compatibilities that remain after this selection process must be achieved to obtain a complete plan. Therefore, we need an additional mechanism to select the order in which they will be implemented. The order determines how the topology of the behavior network evolves, a factor that directly influences the effort required to implement successive compatibilities.

Let us give an example of the OR compatibility selection heuristic. The graph describing value adjacencies can present branching (as in the case of value $s(3n)$ for the WFPC-STATE (Figure 4-3). The current HST model presents this situation for all the state variables associated with the two scientific instruments, and for the telescope pointing state. The topology of each of these graphs, however, grants the existence of a single acyclical path for each ordered pair of values. The existing interactions between the durations of sequences of values and the compatibility temporal relations in the HST model assure that the minimal length path correctly coordinates on time with other state variables. The heuristic therefore consists of a table, indexed by a pair <*branching–value*, *destination–value*>, where each entry represents the first value encountered in the acyclical path starting from the branching value and reaching the destination value.

For another example, let us consider the communication of data to Earth through the fast communication link (1Mb-link); this process requires locking the telescope onto one of the two TDRSS satellites. To minimize idle time, an heuristic chooses the satellite that allows scheduling of the communication operation as soon as possible after the corresponding exposure. The selection considers how the visibility windows of the target required by the exposure overlap with those of the two alternative satellites, together with the kind of synchronization between expose and communicate (e.g., *before*, *contains*).

The mechanism that selects the order of compatibility implementations gives priority to compatibilities that relate values on the same state variable. Whenever such a compatibility is open, a gap (i.e., a constraint token) exists among two values on a state variable. The planner continues to extend the sequence of values on the state variable until the gap is closed.

If no compatibility that relates values on the same state variable is open, then the planner selects a compatibility that involves different state variables. The open "cross" compatibilities among each pair of state variables are organized in a separate queue. The selection mechanism chooses a queue and tries to achieve all the contained compatibilities one by one. However, the satisfaction of a cross compatibility might create a gap on a state variable. In this case, the dequeueing of cross compatibilities is suspended until the new gap is closed.

Implementation of a compatibility: The implementation of a compatibility requires the determination of the position of the requested value or sequence in the behavior network.

A fetch operation linearly scans the sequence of tokens associated to the required state variable

in the behavior network; as a result it returns a list of candidate tokens or sequences of tokens. HSTS uses look-ahead temporal constraint propagation to detect inconsistencies. Mechanisms are provided to limit such propagation to a subnetwork of temporal constraints. By relying on the decomposition of the model into interacting state variables, it is possible to associate each cross compatibility with a set of tightly coupled state variables. The look-ahead constraint propagation will limit the search for cycles to these state variables.

Among the alternatives returned by the fetch operation, a heuristic gives priority to the alternative that is temporally closest to the constrained value, given the goal of minimizing the time spent in reconfiguring the telescope.

8. PERFORMANCE RESULTS

We conducted experiments with three models of the HST operating environment of increasing complexity and realism, respectively denoted as SMALL, MEDIUM and LARGE model. All models share a representation of the telescope at the abstract level as a single state variable; they differ with respect to the number of components modeled at the detailed level. The SMALL model consists only of the telescope pointing device and the Wide Field Planetary Camera. The MEDIUM model adds the two state variables for the Faint Object Spectrograph to the previous model, while the BIG model includes also data communication and tape recorder management. The test problem consists of a set of 50 observation programs, each containing a single observation with no user-imposed time constraints. The experiments were run on a TI Explorer II+ with 16 Mbytes of RAM memory.

The data in Table 8-1 give some measures relative to the final executable schedule that was produced with each model. The number of tokens indicates the total number of distinct state variable values that constitute the schedule. The temporal separation constraints are distance constraints that relate two time points on different state variables; their number gives and indication of the amount of synchronization needed to coordinate the evolution of the state variables in the schedule.

With respect to the processing times, notice that since the heuristics that guide the planning search exploit the modularity of the model and the locality of interactions, the average CPU time (excluding garbage collection) spent implementing each required compatibility in the three models remains relatively stable. The total elapsed time (including garbage collection) spent generating an executable schedule for the 50 observations is an acceptable fraction of the time horizon covered by the schedules; this indicates the practicality of the framework in the actual HST operating environment.

With regard to the quality of the schedules, the percentage of time spent taking exposures with respect to the time horizon covered varies between 22% and 25%. This result is comparable with general expectations of telescope use. However further research is needed to represent the full extent of the actual operating constraints (e.g. treatment of availability of electric power) and to devise strategies to generate efficient schedules for problems involving more complex program constraints. To this end, we are currently investigating the integration of global problem space analysis and focusing techniques [19,9,4]. Some initial results in this direction are reported in [20].

Table I Performance Results Times Reported in Hours, Minutes, Seconds, and Fractions of Seconds			
Model	SMALL	MEDIUM	BIG
State Variables	4	6	13
Tokens	587	604	843
Time Points	588	605	716
Temporal Constraints	1296	1328	1474
CPU Time/Observation	11.62	12.25	21.74
CPU Time/ Compatibility	0.29	0.29	0.33
Total CPU time	9:41.00	10:11.50	18:07.00
Total Elapsed Time	1:08:36.00	1:13:16.00	2:34:07.00
Schedule Horizon	41:37:20.00	54:25:46.00	52:44:41.0

Table 8-1: Performance results

9. CONCLUSIONS

To efficiently operate complex robotics systems, it is necessary to allocate system resources to competing user requests while coordinating system reconfiguration activities. HSTS addresses both concerns within an integrated planning and scheduling architecture. HSTS has been applied to the Hubble Space Telescope observation scheduling domain. Its modeling capabilities allow explicit representation of the various system components and their interactions over time; this leads naturally to a framework for incrementally addressing complex problems, through development of solutions to a series of increasingly more realistic scenarios. The structural characteristics of the model can be exploited by heuristics that guide the coordinated planning processes; this allows the design of efficient planning and scheduling algorithms operating at multiple levels of abstraction. Finally, the evolving schedule is represented as an explicit network of constraints; the flexibility remaining in the final solution (e.g., on the start and duration of various activities) eases the adjustment of a schedule when reacting to unexpected external events.

Acknowledgements

The authors thank Amedeo Cesta, Daniela D'Aloisi, Gilad Amiri and Dhiraj Pathak for their valuable contributions to the HSTS project and Bob Frederking for comments on an earlier draft of this report.

REFERENCES

[1] Muscettola, N., *Planning the Behavior of Dynamical Systems*, Technical Report CMU-RI-TR-90-10, The Robotics Institute, Carnegie Mellon University, 1990.

[2] Waldrop, M., "Will the Hubble Space Telescope Compute?", *Science*, vol. 243, pp. 1437-1439, Mar. 1989.

[3] STScI, *Proposal Instructions for the Hubble Space Telescope*, Technical Report, Space Telescope Science Institute, 1986.

[4] Johnston, M.D., "SPIKE: AI Scheduling for NASA's Hubble Space Telescope", in *Proceedings of the 6th Conference on Artificial Intelligence Applications*, IEEE Computer Society Press, pp. 184-190, 1990.

[5] Baker, K.R., *Introduction to Sequencing and Scheduling*, John Wiley and Sons, New York, 1974.

[6] Panwalker, S.S. and Iskander, W., "A Survey of Scheduling Rules", *Operations Research*, vol. 25, pp. 45-61, 1977.

[7] Fox, M.S. and Smith, S.F., "ISIS: A Knowledge-Based System for Factory Scheduling", *Expert Systems*, vol. 1, no. 1, pp. 25-49, 1984.

[8] Smith, S.F., Ow, P.S., Potvin, J.Y., Muscettola, N. and Matthys, D., "An Integrated Framework for Generating and Revising Factory Schedules", *Journal of the Operational Research Society*, vol. 41, no. 6, pp. 539-552, 1990.

[9] Sadeh, N. and Fox, M.S., "Variable and Value Ordering Heuristics for Activity-based Job-shop Scheduling", in *Proceedings of the Fourth International Conference on Expert Systems in Production and Operations Management*, Hilton Head Island, S.C., 1990.

[10] Ow, P.S. and Smith, S.F., "Viewing Scheduling as an Opportunistic Problem Solving Process", *Annals of Operations Research*, vol. 12, 1988.

[11] Fikes, R.E., Hart, P.E. and Nilsson, N.J.,"Learning and Executing Generalized Robot Plans", *Artificial Intelligence*, vol. 3, pp. 251-288, 1972.

[12] Wilkins, D.E., *Practical Planning*, Morgan Kaufmann, 1988.

[13] Allen, J. and Koomen, J.A., "Planning Using a Temporal World Model", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Karsruhe, Germany, pp. 741-747, 1983.

[14] Dean, T., Firby, R.J. and Miller, D., "Hierarchical Planning Involving Deadlines, Travel Time, and Resources", *Computational Intelligence*, vol. 4, pp. 381-398, 1988.

[15] Lansky, A., "Localized Event-based Reasoning for Multiagent Domains", *Computational Intelligence*, vol. 4, pp. 319-340, 1988.

[16] Vere, S., "Planning in Time: Windows and Durations for Activities and Goals", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, 1983.

[17] Dean, T.L. and McDermott, D.V., "Temporal Data Base Management", *Artificial Intelligence* vol. 32, pp. 1-55, 1987.

[18] Hendrickson, C. and Au, T. *Project Management for Construction*, Prentice Hall, 1989.

[19] Muscettola, N. and S.F. Smith, "A Probabilistic Framework for Resource-Constrained Multi-Agent Planning", *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Milano, Italy, pp. 1063-1066, 1987.

[20] Smith, S.F and Pathak, D.K., *Balancing Antagonistic Time and Resource Utilization Constraints in Over-Subscribed Scheduling Problems*, Technical Report CMU-RI-TR-91-05, The Robotics Institute, Carnegie Mellon University, 1991.