An AI Planning-based Tool for Scheduling Satellite Nominal Operations

Maria Dolores Rodríguez-Moreno, Daniel Borrajo, and Daniel Meziat

■ Satellite domains are becoming a fashionable area of research within the AI community due to the complexity of the problems that satellite domains need to solve. With the current U.S. and European focus on launching satellites for communication, broadcasting, or localization tasks, among others, the automatic control of these machines becomes an important problem. Many new techniques in both the planning and scheduling fields have been applied successfully, but still much work is left to be done for reliable autonomous architectures.

The purpose of this article is to present CONSAT, a real application that plans and schedules the performance of nominal operations in four satellites during the course of a year for a commercial Spanish satellite company, HISPASAT. For this task, we have used an AI domain-independent planner that solves the planning and scheduling problems in the HISPASAT domain thanks to its capability of representing and handling continuous variables, coding functions to obtain the operators' variable values, and the use of control rules to prune the search. We also abstract the approach in order to generalize it to other domains that need an integrated approach to planning and scheduling.

Complex real-world tasks usually require the combination or integration of tools and techniques from two well-known fields—planning and scheduling (Smith, Frank, and Jónsson 2000). An example of such a task is workflow applications, which require the generation of sequences of activities that define a process in an organization and the assignment of resources (human or material) to these activities (Myers and Berry 1999, R-Moreno and Kearney 2002).[1] Other examples are building aircraft (Drabble, McVey, and Clements 2000) and space mission control (Bensana, Lemaitre, and Verfaillie 1999; Dungan et al. 2001; Johnston 1994; Rabideau et al. 2000).[2] To solve problems in any of these domains, we need to represent the information necessary for efficiently finding good solutions. Real domains require a rich representation formalism to be able to handle activities, time, and resource constraints. Several languages have been defined in the AI planning and scheduling community. The PDDL2.2 language (Edelkamp and Hoffmann 2004) is becoming a standard in the planning field for representing domains and problems. Although PDDL2.2 and other predecessor planning languages can be used to represent this type of real problem, in many cases some assumptions have to be made, and in some cases the problem must be reduced.

Traditionally, these problems were solved using methods that belong to either planning or scheduling. On one hand, deliberative planners embody powerful techniques for reasoning about actions and their effects (Allen, Hendler, and Tate 1990). They try to find plans to achieve a set of goals from an initial state and are good at finding precedences among activities, but they are limited at resource or time reasoning.

On the other hand, scheduling systems allocate available resources to known activities over time to produce schedules that respect temporal relations and resource capacity (Cheng and Smith 1995; Tate, Drabble, and Kirby 1994). They are good at optimizing and assigning time and resources to activities, but they require knowledge about ordered relations among the activities. They can optimize a set of objectives, such as minimizing makespan, minimizing work to be done, maximizing resource allocation, or minimizing cycle time.

Depending on the complexity of the problem, solutions to some problems may require a strict separation between planning and scheduling. A simple approach to doing this is to apply a scheduler to the output of the planner to assign resources and time to each activity (Cesta and Pecora 2003). But in other cases there is an indirect temporal and resource dependency with other states and goals that cannot be taken into account if we separate the two tasks (Garrido and Barber 2001). The simple approach is weak if the scheduler fails to find a solution: if the planner does not receive any feedback from the scheduler, the planner can generate expensive and unsatisfactory solutions.

One way of integrating both tasks within the same tool consists of adding representational and reasoning capabilities to the planner for the resource and temporal information. This has been done in systems such as IxTeT (Ghallab and Laruelle 1994), HSTS (Muscettola and Smith 1997), RealPlan (Srivastava, Kambhampati, and Do 2001), and IPSS (R-Moreno, Oddi, Borrajo, Cesta, and Meziat 2004). In this article, we explore the possibility of using a nonlinear domain-independent planner—PRODIGY (Veloso et al. 1995)—and study the possibility of using it directly for generating solutions to problems requiring both planning and scheduling. PRODIGY does not have an explicit model of time representation or a declarative way for specifying resource requirements or consumption. However, thanks to its capability of representing and handling continuous variables, coding functions to obtain the values of the variables, and the use of control rules to prune the search, we have successfully integrated planning and scheduling in the satellite control domain. Satellite control needs to integrate planning (there are implicit precedence relations among operations in the domain description) and scheduling (for instance, the fuel tanks to use should be specified for some operations) to set up nominal operations that must be performed on the satellites during the year.

Our article is structured as follows. In the next section, we describe the features of the planner. The "Satellite Maintenance Operations" and "Scheduling Knowledge in the Planning Domain" sections introduce the type of operations on satellites that have to be performed during the year and the knowledge represented in the planner. The section on the CONSAT tool presents the tool's functionality and the modules that integrate it. We then show some experimental results and, finally, describe related work and draw some conclusions.

Planning and Scheduling: PRODIGY

The planner we use to schedule the HISPASAT operations is PRODIGY (Veloso et al. 1995), an integrated architecture that has been used in a wide variety of domains. The problem solver is a nonlinear planner that uses a bidirectional means-ends analysis search procedure with full subgoal interleaving (Carbonell et al. 1992). The planning process starts from goals and adds operators to the plan until all goals are satisfied. Although it does not use a language with an explicit representation of resources or temporal information, it is able to handle some scheduling reasoning thanks to its capability to (1) represent infinite types (numeric variables), which are needed to represent information about time and resources; (2) define functions that obtain the values of variables in preconditions of operators so that values can be constrained; and (3) use control rules (heuristics) to prune the search, which makes the overall problem-solving process efficient. Figure 1 shows the types of knowledge needed by the planner.

The reasons we chose this particular planner are manifold. Among them, we can highlight definition and handling of quality metrics, explicit definition of control rules, flexibility to define new search behaviors, and explicit rationale of search decisions made through the expansion of the search tree. The domain theory contains all the actions represented by operators. The language for describing PRODIGY domain theory, called PDL4.0 (Carbonell et al. 1992), is based on an augmentation of the STRIPS representation originally proposed by Richard Fikes and Nils Nilsson (Files and Nilsson 1971). In the STRIPS representation, a world state is represented by a conjunction of grounded literals that are true on that state. An operator consists of preconditions (conditions that must be true to execute the action), and postconditions or effects (composed of an add list and a delete list). The add list specifies the set of literals that are true in the resulting state after applying the operators, while the delete list specifies the set of literals that are no longer true after the execution of the action. Because this representation is quite restrictive, it has been extended with disjunctive preconditions, conditional effects, and universally quantified preconditions and effects (Carbonell et al. 1992) resulting in a similar language to ADL (Pednault 1989).

Figure 2 shows an example of an operator in the PDL4.0 syntax in the HISPASAT domain. The operator represents the action that main-

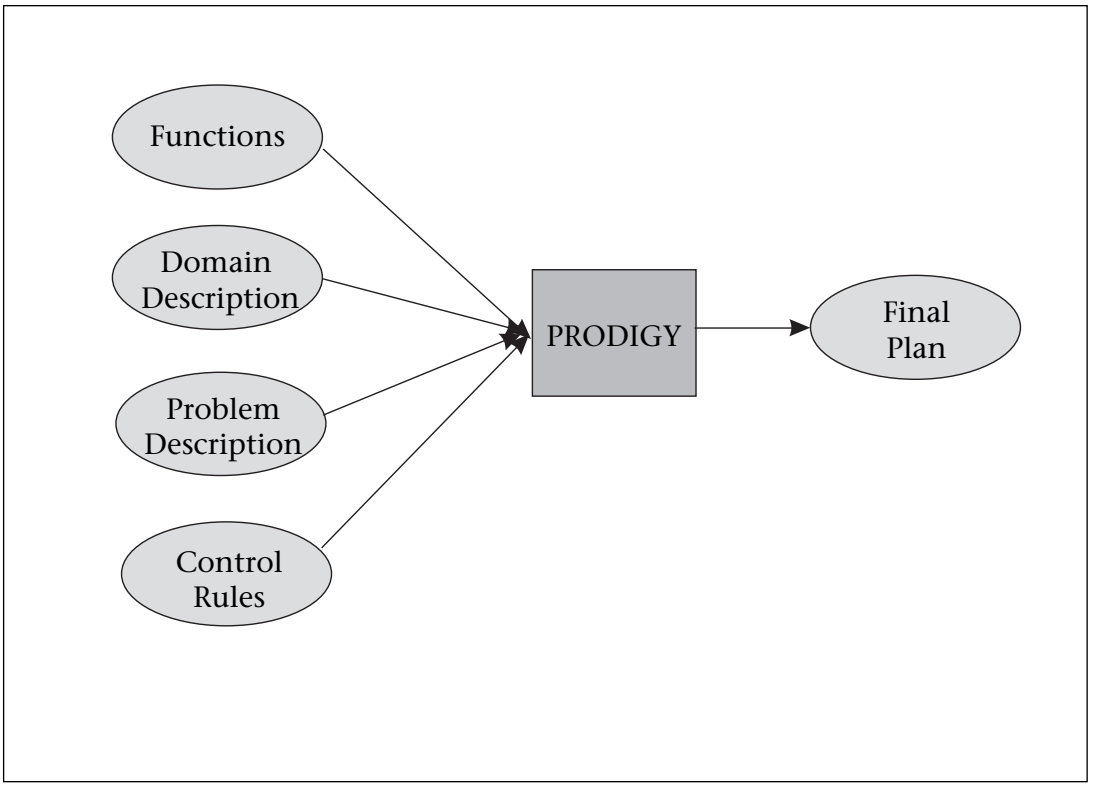

Figure 1. The Inputs and Output of the PRODIGY Planner.

tains the satellite's sensor position. It is performed during the spring and autumn equinoxes.

The symbols within the angle brackets (< >) are variables that are instantiated during problem solving. This operator has two preconditions: *(equinox-spring <d>)* and *(no-maneuver <s> <t> <d0>)* and just one add effect *(ires-transitioned <s>)*. As the variables <*d*> and <*d*0> are numbers, we need to use the *gen-from-pred* PRODIGY function to constrain the values that the numeric variable DATE can have. The *gen-from-pred* function generates a list of values to be possible bindings for the corresponding variable by using the information of the current state referred to as the *no-maneuver* and *equinox-spring* literals. That is, this function permits encoding the functions *d*0 = *no-maneuver*(*s, t*) and *d* = *equinox-spring*(*d*0).

The second input to the planner is the problem to be solved, described in terms of an initial state and a set of goals to be achieved. As a result, PRODIGY generates a plan with the sequence of operators that achieves a state (from the initial state) that satisfies the goals. More importantly, given that we represent some temporal and resource information and constraints within the operators, the plan that PRODIGY generates also takes into consideration the temporal constraints among the operators. The obtained plan does not consider any optimization with respect to resource use or availability, given that for the HISPASAT domain it is enough to find a plan. Any schedule that fulfils the temporal and resource constraints is a valid one. However, the planner could plan for good-quality solutions according to some criteria using the QPRODIGY version described by Daniel Borrajo, Sira Vegas, and Manuela Veloso (2001).

When there is more than one decision to be made at a search decision point, the third input to the planner—the control knowledge (declaratively expressed as control rules)—can guide the problem solver to the correct branch of the search tree. Other planners such as TALplanner (Kvarnstrom and Doherty 2001) have followed a similar approach, although they use a less declarative definition of control knowledge expressed in a form of temporal logic. There are three types of rules: selection, preference, and rejection. One can use the rules to choose an operator, a binding, or a goal or to decide whether to apply an operator or continue subgoaling. Figure 3 shows an example of a control rule to select a binding. From all the batteries that can be used for reconditioning, this control rule selects the battery that is currently unloaded.

```
(OPERATOR  IRES-TRANSITION
  (preconds
    ((<s>  SATELLITE)
     (<t>  TIMES)
     (<d0>  (and DATE (gen-from-pred (no-maneuver <s> <t> <d0>))))
     (<d>   (and DATE (gen-from-pred (equinox-spring <d>)))))
    (and (no-manoeuvre <s> <t> <d0>)
         (equinox-spring <d>)))
  (effects
    ((add (ires-transitioned <s> )))))
```

Figure 2. An Example of a PRODIGY Operator in the HISPASAT Domain.

```
   (Control-rule SELECT-BATTERY
     (if (and (current-goal (loaded-battery <s> <batt>))
              (current-ops (BATTERY-RECONDITIONING))
              (true-in-state (unloaded-battery <s> <batt1>))))
     (then select bindings ((<batt> . <batt1>))))
```

Figure 3. A Control Rule Example that Decides What Battery to Use.

Satellite Maintenance Operations

This section provides an overview of the HISPASAT ground-scheduling operations for its four satellites. HISPASAT is a Spanish multimission system in charge of satisfying national communication needs. It also supplies capacity for digital TV in Europe, America, and North Africa; TV image, radio, and other signals; and special communications for defence purposes. It is the first European satellite system to offer transatlantic capacity for simultaneous coverage between South and North America.

Every maintenance operation in orbit must have explicit engineering instructions. These instructions provide a guide for technicians to consider the work accomplished. The operations engineering group generates this documentation every year by hand and on paper. Later, this documentation is revised and verified.

Due to the increasing number of satellites (there are now five, 1A, 1B, 1C, 1D, and Amazonas, and one planned for the future, called 1E), a program that generates and validates the schedule of operations for engineering support was needed.

In the HISPASAT domain, a special type of operation—the maneuver operation—plays an important role in the scheduling of the rest of the operations. Maneuver operations are used to correctly position the satellite in its orbit. The operations must be executed according to a rigid timetable: every two weeks (on Monday or Tuesday depending if it is summer or winter) in a specific hour, which is given by an external software tool. These operations can be moved ahead only if interference (by the sun or by the moon) occurs. Interference causes incorrect measurment of the satellite orbit, so the sensors affected by interference must be masked using adequate operations. Once these operations are scheduled, the operations related to the use of tanks or batteries are set, always in weeks without maneuvers.

Finally there is a small set of operations that depend only on the last time they were performed. In these cases, just the data of the last operation of their same type is needed.

We have used six ways of eliciting the knowledge that we describe in more detail later in this article: (1) standard written sources, (2) open interviews, (3) structured interviews, (4) questionnaires, (5) the documentation generated for each satellite in previous years, and (6) the satellite operations manuals.

Operators

The first step for defining the domain consists of identifying the operators and the types of objects that are needed in the domain (for declaring the type of each operator variable). Types can be defined and structured in a hierarchy. A special kind of type, the infinite type, can be used to represent variables with continuous values, while finite standard types represent nominal types.

In our domain, we have, among others, the following types: SATELLITE TIME, PERCENTAGE, DIRECTION, and DATE. Variables of type SATELLITE instantiate to one of the available satellites. DIRECTION can have the values north or south. TIMES and PERCENTAGE could have been defined as numbers, but we have chosen to declare them as discrete variables because under our domain formalization only a finite number of values for them are considered. Finally, DATE is represented as an infinite type. Figure 4 depicts one of the hundred operators that have been implemented in the HISPASAT domain—the *South-Maneuver* operator, in charge of computing the date when the operation can be performed, keeping in mind that moon blindings (represented by the *moon-blinding* predicate) cannot interfere (within three hours) with the expected *South-Maneuver* date. If interference occurs, the operation will be moved ahead by 24 hours.

The PRODIGY *gen-from-pred* function generates a list of values to be possible bindings for the corresponding variable by using the infor-

```
(OPERATOR SOUTH-MANEUVER
  (preconds
     ((<s> SATELLITE)
      (<t> TIMES)
      (<t1> TIMES)
      (<p> PERCENTAGE)
      (<n> DIRECTION)
      (<d> (and DATE (gen-from-pred (moon-blinding-start <s> <d> <n> <p> <t1>))))
      (<p1> (and PERCENTAGE (over-n <d> greater 40)))
      (<d1> (and DATE (Is-South-Maneuver <d> 3 hours 3 hours)))
      (<start-time> (and DATE (Calculate-start-time <d1> 24 hours)))
      (<end-time> (and DATE (Calculate-end-time <start-time> 3 hours))))
     (south-maneuver <s> <t> <d1>))
  (effects
     ((add (south-m <s> <t>))
      (add (south-man <s> <t> <start-time>)))))
```

Figure 4. Operator Corresponding to the South-Maneuver Task of Table 1.

mation of the current state. In this particular case we use it to generate all the values that the numeric variable DATE can have. DATE represents seconds, since 1900, in GMT. (We used Common Lisp as the programming language for functions because PRODIGY is written in Lisp and this is the way Common Lisp handles time.) The reason to use this format is for efficiency: it is faster to generate the bindings of one date variable instead of generating values for the six usual time-dependent variables (corresponding to the year, month, day, hours, minutes, and seconds). Also, GMT is the reference zone time for HISPASAT. Other similar approaches fix the starting point of the computation, call it time zero, and schedule all the activities from that point (Muscettola 1994; Tate, Drabble, and Kirby 1994; Vere 1983).

The remainder of the functions that appear in figure 4 have been coded for this particular domain. However, since some of them are generic for any domain with temporal restrictions, they can be reused in any such domain, as will be described later on. As an example of domain-dependent functions, *Is-South-Maneuver* generates the date of the maneuver (if there is any) that overlaps within three hours of any moon blinding. If the blinding intensity is over 40 percent, the maneuver must be moved 24 hours ahead (the function *over-n* calculates if the percentage of the moon blinding is over 40).

As examples of domain-independent functions, the *Calculate-start-time* function subtracts 24 hours from a given date (in this case the expected maneuver), and *Calculate-end-time* calculates the end of the operation from the start time and the duration—three hours in this case. The <*end-time*> variable is computed here just to show how to do it when needed to describe temporal constraints to other operators.

We have defined a similar function, *add-time,* that adds some time to a date and also helps to define temporal constraints among the operators as preconditions of them.

There is only one precondition for the operator: *(south-maneuver* <*s*> <*t*> <*d*1>*)*, the date when the *South-Maneuver* is expected to be performed (part of the initial state, as shown in figure 6).

The operator has two effects that belong to the add list; the predicates *south-m* (the goal that we want to achieve), and *south-man,* which adds to the state the date when the *South-Maneuver* must be performed. In case any interference occurs within three hours, the value of the <*start-time*> variable matches the value of the expected maneuver, <*d*1>, moved ahead 24 hours.

We identified three categories of planned operations, according to the flexibility to schedule them (hard versus soft constraints). The representation chosen for each type can be easily generalized for each planning and scheduling domain. The following subsections describe them in more detail.

Operations Driven by External Events That Start or End at a Fixed Time

Some operations, such as moon blindings, sun blindings, or eclipses of the sun by the earth or by the moon, depend on external events,

Name	South-Maneuver
Description	Follows the two weeks keeping cycle, operating the satellite in one of every two weeks
Requirements	On Monday or Tuesday at an hour corresponding to the secular drift. A West and East maneuver must follow it
Constraints	It can not be performed with Moon or Sun Blindings
Secular drift	16/02/2004 22:47:56
Duration	3 hours

Table 1: South-Maneuver Task.

This is a *South-Maneuver* that will be performed during 2004. If there is a moon blinding within three hours of the expected maneuver with intensity over 40, the maneuver has to be moved ahead 24 hours. The end time of the task is calculated by the *calculate-end-time* function.

which constrain when other operations can be performed. Because these are hard constraints, they are represented as preconditions of the operators. Some external (eclipses and blindings) and seasonal (solstices and equinoxes) events are foreseen several weeks before the year starts, so they are represented in the initial state of the problem. In the case of external events, one needs to include the affected satellite and the events' start and end times. For the seasonal events, one just needs to represent the start time of the spring and summer equinoxes and winter and autumn solstices. Examples are the CHANGE-TO-MODE2/4 and the CONF-ADCS operations performed every time a moon blinding occurs or during the sun blinding periods.

These operations set masks on the upper or lower position detectors depending on where the interference occurs (north or south). By doing so, the satellite does not consider the value of these detectors to define its position. Another example is the CPE-MODE operation performed at the spring and autumn equinoxes, switching the heaters on or off to compensate for seasonal changes. As goals, we define the two possible modes (summer and winter) for the CPE operator, so the planner generates the appropriate satellite operations in each period. The operations in this first category do not force the addition of temporal constraints to an operator's effects.

Given Strategy Operations Performed at Specific Dates

Other operations have more flexibility regarding when they can be performed, such as maneuvers, localization campaigns, or battery reconditioning. Because these operations have no hard constraints, we have coded functions in the preconditions to guide the planner in preferring some dates over others, keeping in mind the restrictions imposed by any preconditions. This decision restricts the number of possible solutions that the planner will provide; however, as we said before, in this domain it is enough to generate a feasible solution. After constraining the variables to the corresponding set of values, the planner is still able to obtain valid plans.

Most of the operations in this category are scheduled to take into consideration the date when a *South-Maneuver* was performed. These operations are in charge of the correct satellite orbit positioning, and they are performed every two weeks at a specific time.[3] In the initial state, we define the satellite affected by the secular drift and the date and time of the year when the maneuver will be performed. Table 1 shows the features of a *South-Maneuver*. Given that the maneuvers are forbidden during moon or sun blindings, they have to be moved ahead 24 or 48 hours (in case two moon blindings occur on subsequent days) from the secular drift time. The two maneuvers (*West-Maneuver* and *East-Maneuver*) that follow the first one must also be moved ahead the same number of hours. The rest of the operations in this category start or finish some hours before or after the start or end of the performance of the *South-Maneuver*. Some operations cannot be performed if a maneuver has been scheduled during that week, while others must be performed during, after, or before maneuvers. Others just have to be performed *N* days, weeks, or months since the last time they were performed. Therefore, we had to represent different types of temporal relations among operators.

To represent this knowledge, we needed a problem solver able to express the fact that if a *South-Maneuver* is performed during one day of a week (having in mind moon and sun blindings), other operations, such as BOOST HEATING, which cannot be performed during the same week but are performed the following week, or the CONF-ADCS operation, which must be performed 9 hours after the *East-Maneuver*, are affected. This kind of reasoning is hard to represent in the operator preconditions and is difficult for most current planners to solve. PRODIGY can solve these problems thanks to the coded functions that restrict the value of the BOOST HEATING start-time variable to the week after the *South-Maneuver*, as we will describe later in the article.

Long-Interval Operations

Some operations, such as maintaining the steerable antenna, have significant scheduling flexibility. This operation is performed four times a year. To schedule it, we need to know when the operation was last performed. The operations in this group present the softest

constraints, because they have no other constraints with the rest of the operators; the constraints relate just to the same operation.

For long-interval operations, we need only to represent in the initial state information about the last time the operation was performed in each satellite during the previous year. With respect to the goals, we have to introduce the number of executions of each operator during the year (in the preceding example, four times).

Control Rules

In the HISPASAT domain, we identified the tanks and the batteries as resources. Control rules can help to assign a given resource, such as a tank, to an operation. Each satellite has four tanks. Any of them could be chosen for the swapping operation, but the recommendation of the company that built the satellite is to use a given tank during each period of the year. The control rule in figure 5 prunes the search tree and chooses the tank for that period of the year. It says that if the date is around the autumn equinox, and we can apply the TANK-SWAPPING operator, then we should select the tank NTO1. The same can be said of the batteries: there are two batteries, BATT1 and BATT2, that must be charged twice a year before every eclipse season. So another control rule helps to choose the battery in the right season.

We could have alternatively coded a function to choose the correct tank for the TANK-SWAPPING operation in the precondition of the operator. However, specifying this type of preference as control rules provides more flexibility, given that it is easier to decide which ones to use at run time, while the domain theory stays general enough.

Initial State and Goals

For each type of satellite control operation, we need to define the set of goals that must be achieved and the initial conditions that must be set up in order to apply them. Figure 6 shows a small fraction of the initial state, that is, all the events that will occur during the year. For instance, for moon blindings we need to know the satellite affected by the blinding (*1B*), the date in seconds when it will take place (3282941400 = 1/13/2004 01:10:00), the direction (south) and the intensity (45 percent represented by *p-45*).

Finally, the first chronological appearance of moon blinding is represented by *t*1, the second by *t2*, and so on. With respect to the goals, we need to perform a fixed number of maneuvers during the year, which is represented in the goals section of the problem.

```
(Control-rule select-tank-NTO1
   (if (and (current-goal (swapped <s> <tank>))
            (current-ops (TANK-SWAPPING))
            (true-in-state (swapped <s> NTO3))
            (true-in-state (equinox-au <d>))))
   (then select bindings ((<tank> . NTO1))))
```

Figure 5. Control Rule to Select the Tank NTO1 during the Autumn Equinox.

```
(state
  (and
     (moon-blinding-start 1B 3282941400 south p-45 t1)
     (moon-blinding-end 1B 3282945005 south p-45 t1)
     (spring-equinox 3257193600)
     (last-antenna-maintenance 1B 3235965395)
     ...
     (south-maneuver 1B t1 3257107800)
     (south-maneuver 1B t2 3258317400)  ...))
(goal
  (and
     (south-m 1B t1) (south-m 1B t2)... ))
```

Figure 6. Some Goals and Initial Conditions for the 1B Satellite.

The goals require that the *South-Maneuver* be performed a given number of times a year.

The definition of this type of state in PDDL2.2 would require the use of functions, but PDDL2.2 cannot handle a case with two or more numerical variables in the same predicate directly.

Planning Versus Scheduling

One might wonder why we use a planner when the problem seems to be a typical scheduling one: locating different activities that consume some resources in specific time windows. As we've already noted, maneuvers are the main operations. Once they are scheduled, everything else can also be scheduled. But maneuvers are scheduled having in mind the atmospheric conditions (blindings and eclipses). These conditions could be modeled as binary resources with a fixed start and end time. Any scheduler will try to schedule the maneuvers when these resources are not being used. However, this will not be a feasible solution for HISPASAT as maneuvers can be moved only 24 or 48 hours ahead, and in the last case, some operations can be omitted or scheduled and planned in a different way depending on the date that the maneuver should be performed. These decisions are not known a priori, so all activities to be scheduled are not known before running the scheduler, and we need some kind of planning.

As a summary, this domain needed a problem solver able to represent and reason about state changes, symbolic and temporal relations among operators, and goals that have to be achieved. Given these constraints, we selected a problem solver that is able to handle all this: PRODIGY.

Scheduling Knowledge in the Planning Domain

In this section, we provide an overview of the scheduling concepts that we had to face in order to give a solution to the nominal operations of HISPASAT using a planner, since we needed to represent time information and constraints through the Allen primitives (Allen 1984). First, we describe the time aspect of the scheduling. Then, we describe some issues about resource usage in this domain.

Representation of Time Constraints

The time representation of PRODIGY is a discrete model of time, in which all actions are assumed to be instantaneous and uninterruptible. It does not handle reasoning on parallel actions. However, the functions that can be called within the preconditions of the operators when assigning values to variables can add constraints among and within operators. Using them, PRODIGY can handle the seven Allen primitive relations between temporal intervals (Allen 1984) and some quantitative relations (Dechter, Meiri, and Pearl 1991; Mieri 1996).[4]

PDDL2.1 is also a discrete model of time in the levels that consider time representation at level 3 by means of temporal conditions and the effects of durative actions, although in PDDL2.2 there is a representation of continuous time. The specification of pre- and postconditions, and the fact that invariant conditions can be identified, means that it can take into consideration concurrent behavior as long as another action that accesses a variable at the exact point when it is updated by another action is avoided. Conflicts over variables can occur only at the start and end points of actions. In the preconditions, the propositions can be asserted at the start of the interval (the point when the action is applied), at the end of the interval (the point when the final effects of the action are asserted), or over the interval from the start to the end (invariant over the duration of the action). In the effects, the literal can be immediately applied (it happens at the start of the interval) or delayed (it happens at the end of the interval).

On one hand, this representation provides more expressivity for the domain modeler than the PDL4.0 syntax does. But a durative action with the features mentioned above can be subdivided into two STRIPS actions, one for each of the end points of the durative action in case there are no invariant conditions (Long and Fox 2001). If the action specifies invariant conditions, it is necessary to guarantee their truth over the interval in order to avoid conflicts. In PRODIGY this is not a problem due to its serial plan nature, but it must be kept in mind in partial order plans as shown in Coddington (2002). On the other hand, PDDL presents more restrictions than PDL4.0 in representing the Allen primitives because PDDL is not able to assign values to variables through coded functions nor to return a set of values within an interval.

Currently, there is not a standard language with respect to scheduling problems, but the wide extended representation as a constraint satisfaction problem (CSP) makes it very easy to handle time and resources. Each operator is represented with two time points, one for the start time and another for the end time. Each time point is represented as an interval of possible values, so all quantitative and qualitative relations between them can be perfectly represented.

To compare the representations mentioned above, we have grouped the seven Allen primitive relations in five types, and we will show how they are represented (for simplicity, in PRODIGY and PDDL2.2 we have reduced the syntax). We have used the HISPASAT domain-independent functions that table 2 shows. There are some functions that return a value and others that return a finite number of possible values in an interval, so these types of functions are obviously discrete. But, as many values as one needs can be returned, so, in the end, they can be thought as equivalent in some practical sense to a continuous representation at a given granularity level. Also, while in the planning notation, a value is assigned, and it is possible to establish constraints, with infinite quantities it is hard to assign a value (commitment). By contrast, in the CSP representation constraints are established much more easily.

The five categories are explained in the following paragraphs.

The end time of OperatorA occurs before the start time of OperatorB within a range of time in the interval [*a, b*]. The following Allen relations belong to this type: OperatorA *before* OperatorB and OperatorA *meets* OperatorB (where $a = b = 0$). The elapsed time from the end of OperatorA can be a value that can be constrained, in the general case, by an interval [*a, b*]. The limits can be zero or positive numbers. The way this can be represented in PRODIGY is shown in

figure 7 (variables DUR and DUR^0 represent the duration; $ELAPSED^1$ and $ELAPSED^2$ represent the minimal and maximal values of the interval [*a, b*]; and TIME, $TIME^0$, $TIME^1$, and $TIME^2$ represent time units). The function *add-time-in-interval* (see table 2) returns a finite set of possible values in that interval. For instance, if we want to represent that OperatorA is between four and five minutes *before* OperatorB, the call to function *add-time-in-interval* should be: *(add-time-in-interval <d> 4 minutes 5 minutes)*, where *<d>* represents the end time of OperatorA. For simplicity we suppose that the start time of OperatorA is given explicitly in the initial conditions of the problem by the predicate/function *start-A*. The representation using the CSP notation is shown in figure 8.

In the case of the PDDL2.2 syntax, we need to declare four functions that represent the start and end times of OperatorA and OperatorB. By defining them as functions, we can modify their values in the effects and do arithmetic and logical operations in the preconditions. Figure 9 shows how to represent these primitives in PDDL2.2. The main difference with the PRODIGY solution is that in the PDDL case, the start time of process B has to be defined in the initial state in order to be able to test its value in the preconditions of OperatorB. On the contrary, PRODIGY can wait until OperatorA is applied to compute when to start execution of OperatorB.

The start time of OperatorA is the start time of OperatorB. The following Allen relations belong to this type: OperatorA *starts* OperatorB and OperatorA *equals* OperatorB. In a case in which we would like to represent the *equals* relationship, we should also constrain in PRODIGY the end time of the operations. Then, the elapsed time from the start of OperatorA to the end of the operation can be a value that can be constrained by an interval [*a, b*] as in the previous case. The way to represent this is depicted in figure 10, while figure 11 shows the CSP representation.

In PDDL2.2 we can impose the same start time in both operations, and the duration of OperatorB can be greater than the end time of OperatorA minus $ELAPSED^2$ and less than the end time of OperatorA plus $ELAPSED^1$. If $ELAPSED^2$ and $ELAPSED^1$ are equal to zero, the duration of OperatorB is equal to the duration of OperatorA, and then it represents the OperatorA *equals* OperatorB primitive (figure 12).

The end time of OperatorA is the end time of OperatorB. The OperatorA *finishes* OperatorB primitive belongs to this category. The case OperatorA *equals* OperatorB could have also been in this category (where $a = b = 0$). In

Name	Meaning
add-time-in-interval <d> $ELAPSED^1$ $TIME^1$ $ELAPSED^2$ $TIME^2$	returns a set of finite possible values for the variable <d> in the interval ELAPSED1 and $ELAPSED^2$ placed to the right of the value of variable <d> according to a given resolution. $TIME^1$ and $TIME^2$ represent time units, that is: hours, minutes, seconds, etc. $ELAPSED^1$ and $ELAPSED^2$ must be positive numbers.
del-time-in-interval <d> $ELAPSED^1$ $TIME^1$ $ELAPSED^2$ $TIME^2$	returns a set of finite possible values for the variable <d> in the interval $ELAPSED^1$ and $ELAPSED^2$ placed to the right of the value of variable <d>. $TIME^1$ and $TIME^2$ represent time units, that is: hours, minutes, seconds, etc. $ELAPSED^1$ and $ELAPSED^2$ must be positive numbers.
add-time <st> DUR TIME	returns a value that is the sum of the variable <st> plus DUR. TIME specifies the time units, so DUR will be converted to seconds according to the time units passed as a parameter.
del-time <st> DUR TIME	returns a value that is the subtraction of the variable <st> minus DUR. TIME specifies the time units, so DUR will be converted to seconds according to the time units passed as a parameter.

Table 2. PRODIGY Domain-Independent Coded Functions to Handle Time.

```
OperatorA
preconds:  (<start-time> (start-A <start-time>))
           (<end-time> (add-time <start-time> DUR TIME))
effects:  (add (finished-A <end-time>))

OperatorB
preconds:  (<d> (finished-A <d>))
           (<start-time> (add-time-in-interval <d>
                          ELAPSED1 TIME1 ELAPSED2 TIME2))
           (<end-time> (add-time <start-time> DUR0 TIME0))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 7. A PRODIGY Representation of the A before B and A meets B Allen Primitives.

```
estB - lftA <= b
estB - lftA >= a
```

Figure 8. CSP Representation of A before B and A meets B.

```
(:functions (start-A)
            (end-A)
            (start-B)
            (end-B))

(:durative-action OperatorA
 :duration (= ?duration DUR)
 :condition …
 :effect (and (at end (assign (end-A) (+ (start-A) DUR)))))

(:durative-action OperatorB
 :duration (= ?duration DUR0)
 :condition (at start (and  (>= (+ (end-A) ELAPSED1) (start-B))
                            (<= (+ (end-A) ELAPSED2 ) (start-B))))
   :effect ... )
```

Figure 9. A PDDL2.2 Representation of the A before B and A meets B Allen Primitives Using Only the Lower Bound of the Interval.

```
OperatorA
preconds: (<start-time> (start-A <start-time>))
          (<end-time> (add-time <start-time> DUR TIME))
effects: (add (started-A <start-time>))
         (add (finished-A <end-time>))

OperatorB
preconds: (<start-time> (started-A <start-time>))
          (<end-time> (add-time-in-interval <start-time>
                       ELAPSED1 TIME1 ELAPSED2 TIME2))
effects: (add (started-B <start-time>))
         (add (finished-B <end-time>))
```

Figure 10. A Representation of the A starts B and A equals B Allen Primitives in PDL4.0.

```
estA = estB
lftB - lftA >= a
lftB - lftA <= b
```

Figure 11. CSP Representation of A starts B and A equals B.

```
(:durative-action OperatorA
 :duration (= ?duration DUR)
 :condition ...
 :effect (and (at end (assign (end-A) (+ (start-A) DUR))))
 ...)

(:durative-action OperatorB
 :duration (and  (>= ?duration (- (end-A) ELAPSED1))
                 (<= ?duration (+ (end-A) ELAPSED2)))
 :condition  (at start ( = (start-A) (start-B)))
 :effect ... )
```

Figure 12: A Representation of the A starts B and A equals B Allen Primitives in PDDL2.2.

PRODIGY the start time of OperatorB computed from the end of OperatorA can be a value that can be constrained, in the general case, by an interval [*a, b*]. The limits can be zero or positive numbers. The function *del-time-in-interval* (see table 2) returns all possible values in that interval. This is represented in figure 13, and its corresponding CSP representation is shown in figure 14.

In PDDL, we have to impose the condition that the end time of both activities be equal. In the case of the OperatorA *equals* OperatorB primitive, the value of the duration of OperatorB should be equal to the duration of OperatorA. Imposing the ending of both operators at the same time should be done in the condition field. But it is not an easy task to guarantee this condition for most of the state-of-the-art planners.

The start time of OperatorB is in a given range from the start of OperatorA, that is, OperatorA *overlaps* OperatorB. The PRODIGY representation is shown in figure 15 and the CSP representation in figure 16.

The start time of OperatorA occurs after the start time of OperatorB, and its end time occurs before the end time of OperatorB. The Allen relation that belongs to this type is OperatorA *during* OperatorB. The way to represent this last category is depicted in figure 17, where the variables ELAPSED^3 and ELAPSED^4 represent the minimal and maximal values of the interval [*a, b*] and TIME^3 and TIME^4 represent time units. The CSP representation of this primitive is shown in figure 18.

Enriching Time Constraints

The relations between operators described in the previous section are defined within a time window and consider that there is only one instance of these relations in that window.

Because in HISPASAT (as well as in some other real domains) there is more than one time window defined with several instances of the same operators on it, we need to differentiate among different executions of an OperatorA and their corresponding relations to different executions of another OperatorB. Therefore, we can define two predicates for each OperatorX to control the instantiated relations:

index-OperatorX: a pointer to the particular instance of each OperatorX.

last-index-OperatorX: the index of the last instantiation of OperatorX.

Also we need to define one more predicate for each temporal relation between two operators:

used-index-OperatorX-for-OperatorY: an index related to an instantiation of OperatorX in relation to another OperatorY.

So for each of the Allen primitives described in this section we should add the predicates explained above as figure 19 shows. We use a counter to compute the instance that has been used for any of the temporal relationships between operators and a global counter to compute the last value used. Given that this solution requires some modeling effort for humans, we can hide it by building an interface that automatically defines these high-level relations between operators and translates these relations into instances following the notation described above. A quite similar problem happens in CSP, because we need to know all activities a priori that need to be scheduled in the plan.

Representation of Resources Constraints

A resource is a source of supply or support or an available means. There are basically three types of resources (Sherwood et al. 2000), although the name given to each type varies from one author to another:

Type 1 is available when not in use (one user at a time). Examples are physical devices such as a robot arm or a CPU. In ASPEN (Sherwood et al. 2000) a similar type is the *concurrency* resource that must be made available to the activity before resources are reserved. An example would be a telecommunications downlink pass. The telecommunications station must be available before the spacecraft could initiate a downlink.

Type 2 can be used by more than one activity, so a capacity should be defined. It is always available when not in use, and many activities can use different quantities of it. It does not need to be replenished as, for example, solar array power does.

In Type 3 the capacity is diminished after its use, so a capacity should be defined. It may or not be replenished by another activity. Examples are battery energy, memory capacity (which can be replenished) and fuel (which cannot be replenished for satellite missions).

In PRODIGY, as in PDDL2.2, there is no provision for specifying resource requirements or consumption. But resources can be seen as variables that can have associated values through literals that refer to them, that is, as a logical formula. This way of resource representation has the disadvantage of making the search extremely intractable when the number of resources increases, as the results in Srivastava, Kambhampati, and Do (2001) show.

For example, if tank T1 of a satellite has 30 liters of fuel in a given instant, we could represent it as *(has-fuel T1 30)*. Then, in the opera-

```
OperatorA
preconds: (<start-time> (start-A <start-time>))
          (<end-time> (add-time <start-time> DUR TIME))
effects: (add (finished-A <end-time>))

OperatorB
preconds: (<end-time> (finished-A <end-time>))
          (<start-time> (del-time-in-interval <end-time>
                         ELAPSED¹ TIME¹ ELAPSED² TIME²))
effects: (add (started-B <start-time>))
              (add (finished-B <end-time>))
```

Figure 13. A Representation of the A finishes B and A equals B Allen Primitives in PRODIGY.

```
lftA = lftB
```

Figure 14. CSP Representation of the A finishes B and A equals B Allen Primitives.

```
OperatorA
preconds: (<start-time> (start-A <start-time>))
          (<end-time> (add-time <start-time> DUR TIME))
effects: (add (started-A <start-time>))
         (add (finished-A <end-time>))

OperatorB
preconds: (<d> (started-A <d>))
          (<start-time> (add-time-in-interval <d>
                         ELAPSED¹ TIME¹ ELAPSED² TIME²))
          (<end-time> (add-time <start-time> DUR⁰ TIME⁰))
effects: (add (started-B <start-time>))
         (add (finished-B <end-time>))
```

Figure 15. A Representation of the A overlaps B Allen Primitive in PRODIGY.

```
lftB < lftA
estB <= estA - a
estB >= estA - b
```

Figure 16. A overlaps B Allen Primitive Representation Using CSP.

```
OperatorA
preconds: (<start-time> (start-A <start-time>))
          (<end-time> (add-time <start-time> DUR TIME))
effects: (add (started-A <start-time>))
         (add (finished-A <end-time>))

OperatorB
preconds: (<d> (started-A <d>))
          (<d1> (finished-A <d1>))
          (<start-time> (add-time-in-interval <d>
                         ELAPSED¹ TIME¹ ELAPSED² TIME²))
          (<end-time> (del-time-in-interval <d1>
                       ELAPSED³ TIME³  ELAPSED⁴ TIME⁴))
effects: (add (started-B <start-time>))
         (add (finished-B <end-time>))
```

Figure 17. A Representation of the A during B Allen Primitive in PRODIGY.

```
estB <= estA - b
estB >= estA - a
lftB >= lftA + c
lftB <= lftA + d
```

Figure 18. A during B Primitive Using CSP.

```
                        Initial State
    for each OperatorA --> (last-index-opA 0)
                           (used-index-opA-for-opB 0)

    OperatorA
    preconds: (<last-index> (last-index-opA <last-index>))
              (<index> (+ 1 <last-index>))
              ...
    effects: (add (index-opA <index>))
             (del (last-index-opA <last-index>))
             (add (last-index-opA <index>))

    OperatorB
    preconds: (<index> (index-opA <index>))
              (not (used-index-opA-for-opB <index>))
    ...
    effects: (add (used-index-opA-for-opB <index>))
```

Figure 19. A Way of Handling Instances for Any Allen Primitives.

```
OPERATOR-MANEUVER-SOUTH
preconds: (<angle> (position <sat> <angle>)
                   (> <angle> 4))
          (<available> (available-fuel <sat> <available>))
          (<fuel-consumed> (/ <available> <angle>))
          (<new-fuel> (- <available> <fuel-consumed>))
effects: (add (performed-maneuver  <sat>))
         (add (available-fuel <sat> <new-fuel>)))
         (del (available-fuel <sat> <available>)))
```

Figure 20. Representing a Resource Capacity and Consumption Such as Fuel in PRODIGY for the Satellite Domain.

```
(:action Maneuver-S
 :parameters (?sat - satellite)
 :precondition  (> (position ?sat) 4)
 :effect  (at end (performed-maneuver  ?sat))
          (at end (decrease (available-fuel ?sat)
                            (/ (available-fuel ?sat) (position ?sat))))
```

Figure 21. PDDL2.2 Representation of Figure 20 Operator.

tors, we can restrict the set of values that can be assigned to the variable that represents the resource, consume the fuel, or refuel the tank if necessary and possible.

To represent capacity, we can use the scalar quantity model. The capacity constraints of a resource with uniform capacity can be calculated in PRODIGY through a functional expression. For instance, we can define a function (such as *compute-consumed-fuel*) that calculates the fuel consumed in each maneuver as a function of the available fuel and the angle of the satellite with respect to the sun as shown in figure 20.

The value of the angle is obtained from the instantiation of the literal *(position <sat> <angle>)* and must be greater than the value four (4 grades) with respect to the current state. Once it sets the value of variable <*sat*>, it will access the state and set the value of the second argument as the value for variable <*angle*>. Also, the value of the available fuel is obtained from the instantiation of the literal *(available-fuel <sat> <available>)* with respect to the state, for the instantiated variable <*sat*>.

In the effects, the operator adds the grounded literals *(performed-maneuver <sat>)* and *(available-fuel <sat> <new-fuel>)* to the state (in other words, it asserts them as true) and deletes the *(available-fuel <sat> <available>)* literal from the state.

In PDDL2.2, the functions *decrease/increase* present a compact way of handling numeric fluents, more compact than in PRODIGY, but the semantics and the way to use resource consumption is the same, as figure 21 shows.

In the case of binary resources, in PDL4.0 and PDDL2.2 we can model them as predicates that must be available at the beginning of the operation. The operation will remove a resource's availability from the state (we can do that because of the instantaneous representation of actions) so other operations cannot use it, but we need to add an operation that sets the resource free so it can be used again. Figures 22 and 23 show an example of a binary resource as it can be an instrument inside a satellite. This example belongs to the satellite domain of the 2002 International Planning Competition (www.dur.ac.uk/d.p.long/IPC). In this case, scientific instruments in satellites must collect some data as images. In order to observe some data, the instrument inside the satellite must be free. The operation adds to the state that the instrument is busy. In order to use it again, the *Free-Resource* operation will add to the state its availability for other operations (the *(del (busy <instrument>))* or *(not (busy ?i))* grounded literals).

In CSP scheduling, there are many algorithms and heuristics to handle easily binary to multicapacity resources, such as the ones proposed in Cesta and Oddi (2002); Cesta, Oddi, and Smith (1999); and Smith and Cheng (1993).

The Tool: CONSAT

One of the current problems for the deployment of planning technology is the lack of an easy-to-use front end for users and their integration with the current tools used in organizations. In this section we present a graphical modeling and validation tool developed for scheduling the nominal operations for in-orbit control of HISPASAT's satellites (R-Moreno, Borrajo, and Meziat 2002). The CONSAT[5] tool consists of the following subsystems, which will be described in the next subsections:

The User Subsystem is in charge of the control access to the tool and the interaction with the user in order to obtain and manipulate all the data needed for planning and scheduling.

The Reasoner Subsystem: once the input data is introduced, a domain-independent planner is in charge of generating the solution to the problem.

The Generator Subsystem is responsible for maintaining coherence between the two possible representations that the tool offers: *annual* to provide a general overview of the operations and *weekly* for a more detailed view of the hours and resources (if any) involved in the operations. If the user modifies the *weekly* representation, the *annual* representation will be updated automatically. This subsystem also converts the solution to a specific document format type, comparing two different solutions, or generating an HTML version.

Figure 24 shows a high-level view of the architecture and the modules that it comprises. The first and the third subsystems interact with the user. The second subsystem interacts with the other two and is in charge of the solution generation.

The User Subsystem

The User Subsystem is in charge of data introduction. The control engineers are the ones who interact with this subsystem. As we have seen before, the initial state and goals refer to data about external events and some operations. Some of this data, such as the time in seconds that most operators and literals need as arguments, is difficult for a user to provide directly. Therefore, the user can specify data in the current usual way through the User Subsystem, and the subsystem automatically translates it into the internal model.

```
Take-Data
preconds: (belong  <instrument> <sat>)
          (not (busy <instrument>))
effects: (add (busy  <instrument>))
         (add (have-image <mode>  <direction>))
Free-Resource
preconds: (busy <instrument>)
effects: (del (busy  <instrument>))
```

Figure 22. An Example in PRODIGY of How to Represent a Binary Resource.

```
(:action Take-Data
 :precondition (and (belong  ?i ?sat)
                    (not (busy ?i))
 :effect: (and (busy  ?i)
                    (have-image ?m ?d)))

(:action Free-Resource
 :precondition (busy ?i)
 :effect:  (not (busy  ?i)))
```

Figure 23. The Example of Figure 22 in PDDL2.2 Syntax.

This subsystem provides the following more specific functionality: In the case of external and *South-Maneuvers*, the events are known in advance by means of specific supporting engineering tools that predict, according to some parameters, when the events will occur and the hour when the *South-Maneuver* can be performed every day of the year. These software tools generate the data in a set of ASCII files. The subsystem can also import these files and detect the correct file format.

Nowadays, the engineers represent this data in a specific document and schedule the operations according to it. Every time a new modification is done, the engineer in charge of it must sign the document. Therefore this subsystem also controls the user access to register who is creating, revising, modifying, or verifying the schedule.

Related to the third type of operations, which require data from the last time the operation was performed, the engineers must find the information in the previous year's document. If the tool was previously used, this data is available to the User Subsystem and automatically reused.

Finally, the User Subsystem assists the user in validating the introduced data, thus avoiding possible user inconsistencies, such as trying to swap the same tank twice in the same period of the year or performing an operation in an illegal period.[6] Figure 25 shows one of the inter-

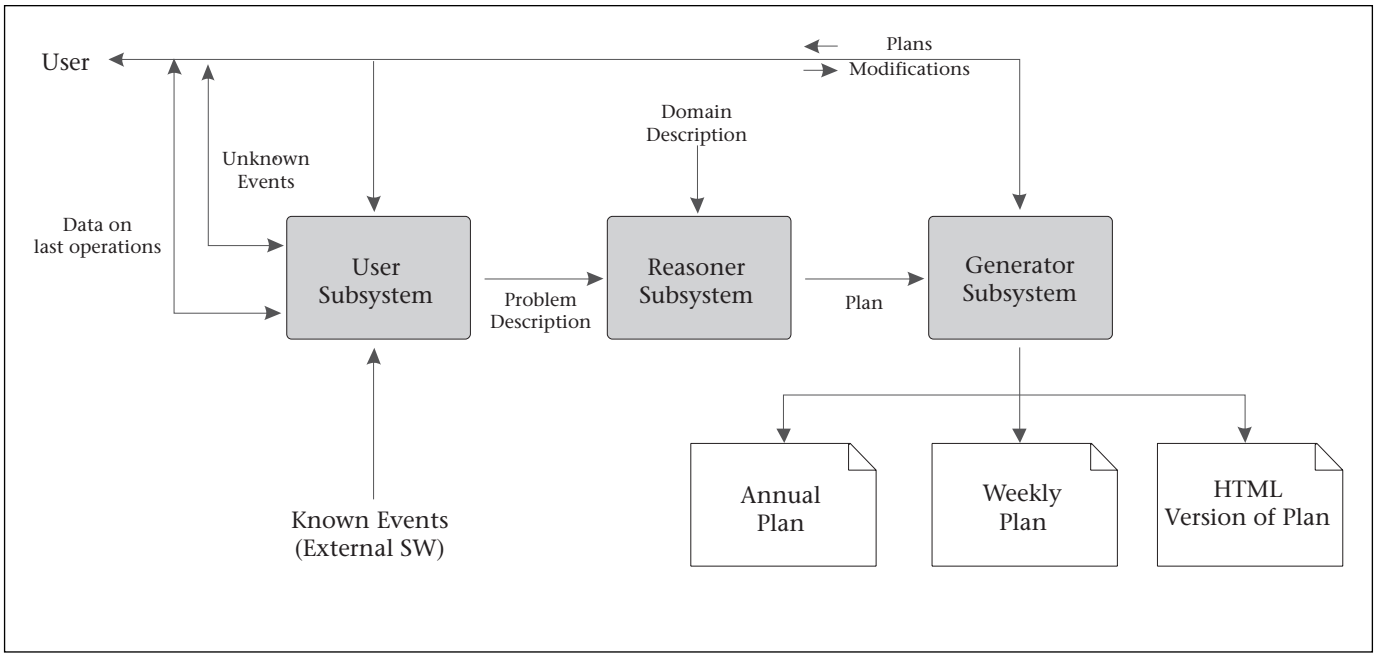

Figure 24. Architecture of the Planning Tool.

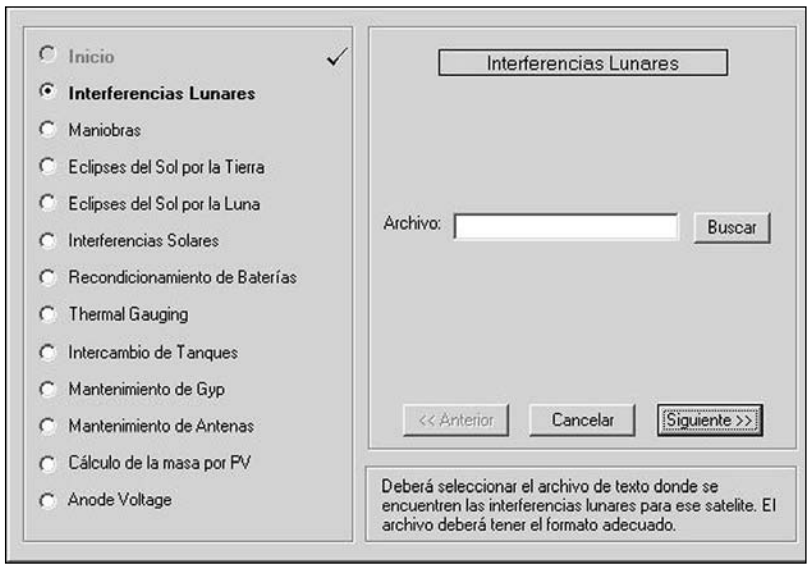

Figure 25. One of the Windows of the User Subsystem Interface.

face windows (written in Spanish due to engineers' requirements) of the User Subsystem. It shows on the left part of the window an ordered list of operations that require user inputs. On the right side, the system asks for specific data for each operation, such as a file (as shown in the figure) or the last performed operation if it is not available in the database. The user must complete each step on the left part of the window in order to obtain an annual schedule.

The Reasoner Subsystem

The User Subsystem translates all this information into a suitable format for input to the Reasoner Subsystem. This system is composed of the AI planner explained previously. It is in charge of the plan generation, with temporal and resource reasoning. It generates a problem file with all the information provided by the user. This file is given to the planner as an input, together with the domain and control knowledge files, allows the planner to run. The planner output is saved in another ASCII file that will be manipulated to generate the input to the Generator Subsystem.

The Generator Subsystem

Currently, once the engineers know every external event and have represented the events in a document, the laborious task of scheduling every operation starts. The engineers generate two types of documents: (1) a document that provides an overview of all the operations performed each day of the year; and (2) a document that represents in more detail each operation's duration, type of maneuver, and any resources, such as batteries, tanks, and so on, affected by the operation.

The weekly representation is generated every week and takes into consideration the annual representation (shown in figure 26). The problem of maintaining two documents relates to the incongruencies between them. Also, these documents are generated by the engineers of

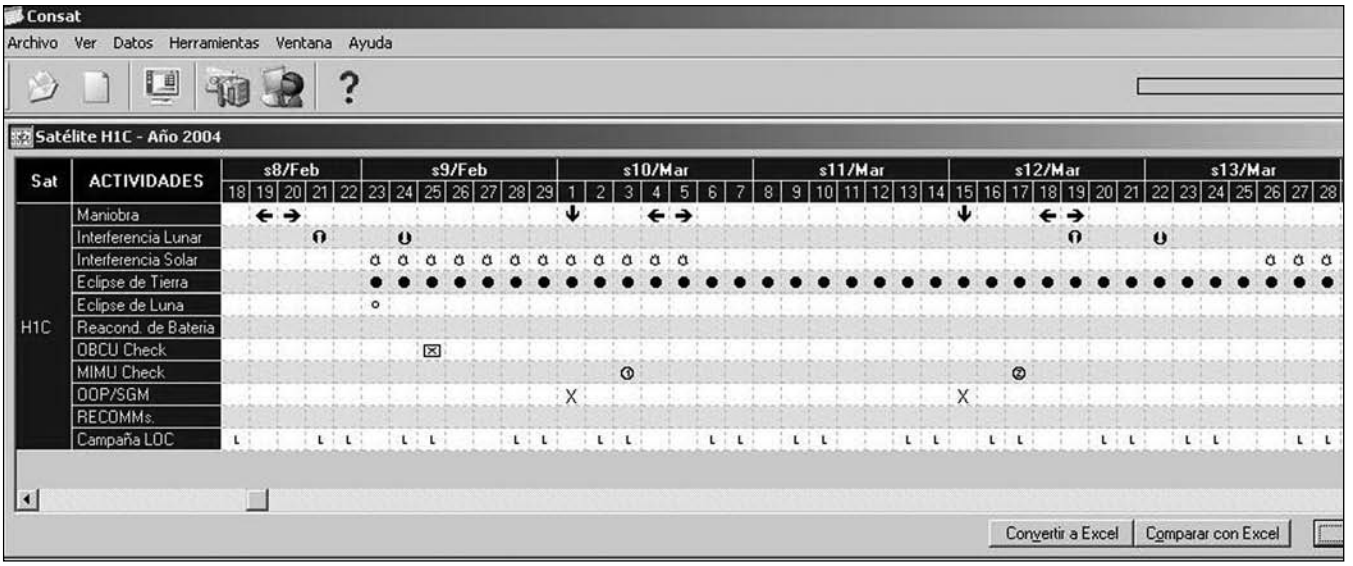

Figure 26. The Generator Subsystem Interface: Annual Representation.

Satellite	Time (mn)	Goals	Literals in the initial state	Nodes	Ops. in solution
1A	9.13	215	93	1837	411
1B	8.61	247	91	1947	439
1C	2	221	160	1269	303

Table 3. Results Obtained to Achieve a Plan for Each Satellite.

the headquarters at Arganda (Madrid) and sent to other backup centers in Spain and South America. In case any problem arises at the headquarters, one of those other backup centers must continue performing the scheduled operations. Therefore, every time a change on a schedule occurs, it must be sent to the rest of the backup centers.

The Generator Subsystem guarantees the consistency of the two representations. The user can easily modify the results obtained by the planner by just dragging and dropping the symbols in the table of the annual or weekly representations. The user can also compare two solutions and analyze the differences between them; convert the results to the document format that the engineers use in daily operation; and generate the solution in HTML for use at the other centers.

Experimental Results

We have done some experiments to show the performance of the tool developed for HISPASAT on real data that are shown in table 3. All these results have been obtained using a Pentium III 800 MHz processor and 256 megabytes of memory under Windows. The number of operators in the HISPASAT domain is 100, the number of control rules is 10, and there are a total of 50 coded functions, 25 percent of them domain independent.

To compare it with humans, the time devoted to preparing the annual representation for the three satellites is 40 hours a year by one person, basically 35 hours to elaborate it and 5 hours for verification. For the weekly representation, engineers devote one hour and a half a week, and generally not all the changes made in the weekly representation are reflected in the annual representation.

With respect to validation of generated plans, all of the plans are valid. We analyzed differences with the plans provided by human experts, and the results were that the plans generated by the tool agree at least 90 percent of the time with the plans generated by the engineering team. The main differences fall on last-time operation changes due to the satellites' needs or coincidence of some operations with holidays that force the operations to be moved to other dates. This last discrepancy will be considered in future versions of the software. CONSAT is actually under the validation phase.

Related Work

Several planning systems have addressed applications of planning with resources and time considerations. The planner NONLIN+ (Tate and Whiter 1984) maintains information about the duration of the activities and represents limited resources.

In SIPE (Wilkins 1988) the use of resouces can be declared in the operators. DEVISER (Vere 1983) could also handle consumable resources. These planners have been designed with the purpose of explicitly handling some types of resources. In the IPP planner (Koehler et al. 1997), based on Graphplan (Blum and Furst 1997), the search algorithm has been modified to combine the ADL search algorithm to handle logical goals together with interval arithmetic to handle resource goals. Realplan (Srivastava, Kambhampati, and Do 2001), also based on Graphplan, decouples the logical reasoning from resource reasoning thanks to the CSP formulation that helps the planner in resource allocation. This is the main difference with PRODIGY, which was designed as a general planning and learning system rather than as a scheduler. But this is not a disadvantage for representing this type of knowledge and reasoning about it without modifying the search algorithm (as explained in the "Satellite Maintenance Operations" and "Scheduling Knowledge in the Planning Domain" sections).

Other current approaches have integrated planning and scheduling in the same system as HSTS (Muscettola 1994) by instantiating state variables into a temporal database. These state variables are very different from predicate logic formulae used by PRODIGY, although they are merely a convenient shorthand for logic, leading to no loss in expressivity. Also the O-PLAN2 (Tate, Drabble, and Kirby 1994) integrates planning, scheduling, execution, and monitoring within a system that also uses an activity-based plan representation. Based on HTN, it requires a lot of hand-coding of domain operations and refinement of these operations in hierarchical levels. Other systems are temporal-based planners, such as IxTeT (Ghallab and Laruelle 1994). It is a least commitment planner that uses a graph-based algorithm for detecting resource conflicts between parallel actions. The time logic relies on a restricted interval algebra represented by time points. Time points are seen as symbolic variables where temporal constraints can be posted. The world is described by a set of multivalued domain attributes temporally qualified into instantaneous events and persistent assertions and a set of resource attributes. SPIKE (Jónsson et al. 2000) was initially used for science operations for the Hubble space telescope ground system, but then it was adapted to schedule a variety of astronomical scheduling problems. It has faced the problem using CS techniques.

Magpen (Ai-Chang et al. 2003) is part of the Mars Exploration Rover mission. It is a system that merges ideas from CS as propagation and consistency checking and planning. The planner system called EUROPA (Frank, Jónsson, and Morris 2000) is an evolution of the Deep Space One planner (Muscettola 1994). The partial plan that it builds consists of a set of intervals connected by constraints. This plan is modified using search-based methods until the plan is a valid one.

All these systems differ from our approach in that they had to (re-)implement the planners to handle scheduling information, while we have used an "old" planner with a powerful representation of domains and defined a set of simple time-handling functions that introduce time management in an intuitive way almost equivalent to the expressive power of other tools.

ASPEN (Rabideau et al. 1999) uses an AI planner with a richer representation in activities so the user can easily provide start times, end times, duration, and use of one or more resources. Some algorithms this planner uses to solve the problem belong to the scheduling area (such as the schedule iterative repair algorithm). ASPEN is part of CASPER (continuous activity scheduling planning execution and replanning) (Rabideau et al. 2000). It provides functionality to work in dynamic environments, a requirement for most NASA projects (Chien et al. 1999; Chien et al. 2002; Chien et al. 2003; Estlin et al. 1999; Willis, Rabideau, and Wilklow 1999). Our planner uses specific planning algorithms that can handle a subset of the scheduling problem, sufficient for solving tasks in the HISPASAT domain. We do not need replanning, operations monitoring, or execution because these operations are not as critical as on-board missions, and planning execution time is not a high priority. The flexibility to schedule the operations is one of the main features of this domain as opposed to on-board missions where maximizing the weighted sum of the scheduled observations is one of the main issues.

A quite similar domain as the one presented in this article is shown in the Ground Processing Scheduling System (GPSS) (Deale et al. 1994). The way they face the problem differs from ours in several aspects. First, GPSS takes as inputs the set of precedence relations between tasks. It defines four types of temporal constraints with intervals between two tasks: activity-A *finishes* be-

fore the *start* of activity-B, the *start* time of activity-A is equal to the *start* time of activity-B, the *finish* time of activity-A coincides with the *finish* time of activity-B, and the *start* time of activity-A coincides with the *finish* time of activity-B. Second, resources are modeled as classes separately with an initial capacity. Third, to handle changes produced by tasks, tasks are represented as attributes instead of as predicate logic as in our approach. Fourth, a solution is a schedule where each task has a start and end time, a resource assignment for every resource that the activity consumes where all temporal constraints are satisfied. When looking for solutions, GPSS searches through the space of all possible schedules, looking for better schedules thanks to the defined cost functions based on expert heuristics.

In our approach we have as inputs the domain theory and the problem definition. For our domain, the four precedence relations between tasks are not enough, as we need other Allen primitives such as *during* or *overlaps,* and we found it easier to model these relationships between activities inside the domain theory. We could also add intervals between tasks thanks to the coded functions. In relation to resources, these are part of the causal reasoning, so PRODIGY, as most planners, considers discrete resources like robots, fuel, or batteries as logical predicates. This causes the search space to become huge when the number of resources increases. As experimental results showed in Srivastava, Kambhampati, and Do (2001), this strategy severely curtails the scale-up of existing planners. Given that the HISPASAT domain does not deal with a high number of resources, it does not affect PRODIGY performance. With respect to how to represent changes, in PRODIGY as in any other classical planner, an operator consists of preconditions and effects. Actions have well-defined start and end times and resource assignments for each activity. The obtained plan does not consider any optimization with respect to resource use or availability, given that for HISPASAT it is enough to find a plan. However, as mentioned before, we could also reason about quality-oriented plans according to some criteria using QPRODIGY (Borrajo, Vegas, and Veloso 2001).

Conclusions and Future Work

In this article, we have presented a tool that fully integrates planning and scheduling for the nominal operations that need to be performed in three satellites during the course of a year for a commercial satellite company. Before the software development, the engineering group generated by hand and on paper the operations that have to be done during the week and year. There were many incongruencies between the two types of representation used (weekly and annual), and document modification was a tedious task.

The tool not only saves users a lot of time due to its capability of importing files of any type, it presents the results in a table that can be easily modified by just dragging and dropping, generates more than one solution, shows the differences between any two solutions, and generates the results in their internal format or in HTML. But there is also a high-level description language that is used to specify satellite problems (domain-dependent knowledge) and planning and scheduling problems (domain independent knowledge) and that makes it easy to maintain the tool and to add the correspondent operations when new satellites are added.

We want to explore the possibility of adding more satellites, so that CONSAT could define or delete new operations, and be adapted to new satellites as Magpen (Ai-Chang et al. 2003) is adapted to different missions. This would allow us to study the scalability of the approach for dealing within the planner with planning, temporal, and resource reasoning. It would also be interesting to integrate a monitoring module to execute the scheduled operations obtained by the planner and to replan in case of changes. Finally, we would like to explore the tool's generality by using the satellite domains' temporal independent knowledge for coding other domains that require planning and scheduling.

Acknowledgements

We want to thank the entire HISPASAT engineering team for their help and collaboration during development of this project—especially Arseliano Vega, Pedro Luis Molinero, and Jose Luis Novillo. This work was partially funded by the CICYT project TAP1999-0535-C02-02 and TIC2002-04146-C05-05.

Notes

1. See also the PLANET Workflow Management TCU Road Map (http://scalab.uc3m.es/~dborrajo/planet/wmtcu).

2. See also the PLANET Aerospace TCU Road Map (http://pst.ip.rm.cnr.it/en/events/planet).

3. The hour corresponding to the secular drift direction, that is, an hour when it is possible to position the satellite, having in mind different parameters.

4. Because it cannot return infinite possible values for variables and it does not reason about variables that represent intervals, it cannot handle all quantitative relations as CSP techniques do.

5. CONSAT stands for satellite control in Spanish.

6. A legal period in TANK SWAPPING is a date around equinox, for instance.

References

Ai Chang, M.; Bresina, J.; Charesty, L.; Hsu, J.; Jónsson, A.; Kanefsky, B.; Maldaguey, P.; Morris, P.; Rajan, K.; and Yglesias, J. 2003. Mapgen Planner: Mixed-initiative Activity Planning for the Mars Exploration Rover Mission. Paper presented at the Seventh International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS). Naras, Japan, 19–23 May.

Allen, J. 1984. Towards a General Theory of Action and Time. *Artificial Intelligence Journal* 23(2): 123–154.

Allen, J. F.; Hendler, B.; and Tate, A. 1990. *Readings in Planning*. San Francisco: Morgan Kaufmann Publishers.

Bensana, E., Lemaitre, M., and Verfaillie, G. 1999. Earth Observation Satellite Management. *Constraints: An International Journal* 4(3): 293–299.

Blum, A.; and Furst, M. 1997. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence Journal* 90(1–2): 281–300.

Borrajo, D.; Vegas, S.; and Veloso, M. 2001. Quality-Based Learning for Planning. Paper Presented at the IJCAI'01 Workshop on Planning with Resources, Seattle, WA, 6 August.

Carbonell, J. G.; Blythe, J.; Etzioni, O.; Gil, Y.; Joseph, R.; Kahn, D.; Knoblock, C.; Minton, S.; Pérez, A.; Reilly, S.; Veloso, M.; and Wang, X. 1992. PRODIGY4.0: The Manual and Tutorial. Tech. Rep. CMU-CSP-92-150, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Cesta, A.; and Oddi, A. 2002. Algorithms for Dynamic Management of Temporal Constraint Networks. Tech. Rep., Italian National Research Council (ISTC-CNR), Rome, Italy.

Cesta, A.; Oddi, A.; and Smith, S. F. 1999. Greedy Algorithms for the Multi-Capacitated Metric Scheduling Problem. In *Proceedings 1999 European Conference on Planning. Lecture Notes in Computer Science*. Berlin: Springer-Verlag.

Cesta, A.: and Pecora, F. 2003. The RoboCare Project: Multi-Agent Systems for the Care of the Elderly. *European Research Consortium for Informatics and Mathematics (ERCIM) News* No. 53 (2003).

Cheng, C. C.: and Smith, S. F. 1995. Applying Constraint Satisfaction Techniques to Job Shop Scheduling. Tech. Rep. CMU-RI-TR-95-03, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Chien, S.; Rabideau, G.; Willis, J.; and Mann, T. 1999. Automating Planning and Scheduling of Shuttle Payload Operations. *Artificial Intelligence Journal* 114(1–2): 239–255.

Chien, S.; Rabideau, G.; Willis, J.; and Mann, T. 2002. The RADARSAT-MAMM Automated Mission Planner. *AI Magazine* 23(2): 25–36.

Chien, S.; Sherwood, R; Tran, D.; Castano, R.; Cichy, B.; Davies, A.; Rabideau, G.; Tang, N.; Burl, M.; Mandl, D.; Frye, S.; Hengemihle, J.; DAgostino, J.; Bote, R.; Trout, B.; Shulman, S.; Ungar, S.; Van-Gaasbeck, J.; Boyer, D.; Griffin, M.; Burke, H.; Greeley, R.; Doggett, T.; Williams, K.; Baker, V.; and Dohm, J. 2003. Autonomous Science on the EO-1 Mission. Paper presented at the Seventh International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS). Naras, Japan, 19–23 May.

Coddington, A. 2002. A Continuous Planning Framework with Durative Actions. In *Proceedings of the Ninth International Symposium on Temporal Representation and Reasoning (TIME-2002),* 108–118. Los Alamitos, CA: IEEE Computer Society.

Currie, K.; and Tate, A. O-Plan: The Open Planning Architecture. 1991. *Artificial Intelligence Journal* 52(1): 49–86.

Deale, M.; Yvanovich, M.; Schnitzius, D.; Kautz, D.; Carpenter, M., Zweben, M.; Davis, G.; and Daun, B. 1994. The Space Shuttle Ground Processing Scheduling System., 423–449. In *Intelligent Scheduling,* ed. Mark Fox and M. Zwebem. San Francisco: Morgan Kaufman, Publishers.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence Journal* 49(1–3): 61–95.

Drabble, B.; McVey, C.; and Clements, D. 2000. Agile Aircraft Manufacturing and Assembly. Paper Presented at the Symposium on Planning, Scheduling, and Control for Aerospace, World Congress on Automation (WAC-2000). Wailea, Hawaii, June 11–16.

Dungan, J.; Frank, J.; Jónsson, A.; Morris, R.; and Smith, D. 2001. Planning and Scheduling for Fleets of Earth Observing Satellites. Paper presented at the Sixth International Symposium on AI, Robotics, and Automation in Space (i-SAIRAS 2001). Saint-Hubert, Quebec, June 18–22.

Edelkamp, S.; and Hoffmann, J. 2004. pddl2.2: The Language for the Classical Part of IPC-4 at the Fourth International Planning Competition. Technical Report No. 195. Frieburg University Institute of Computer Science, Frieburg, Germany.

Estlin, T.; Rabideau, G.; Mutz, D.; and Chien, S. 1999. Using Continuous Planning Techniques to Coordinate Multiple Rovers. Paper Presented at the IJCAI-99 Workshop on Scheduling and Planning Meet Real-time Monitoring in a Dynamic and Uncertain World. Stockholm, Sweden.

Fikes, R.; and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence Journal* 2(3–4), 189–208.

Frank, J.; Jónsson, A.; and Morris, P. 2000. On Reformulating Planning as Dynamic Constraint Satisfaction. In *Proceedings of the Fourth Symposium on Abstraction, Reformulation and Approximation (SARA)*. Berlin: Springer-Verlag.

Garrido, A.; and Barber, F. 2001. Integrating Planning and Scheduling. *Applied Artificial Intelligence* 15(5): 471–491.

Ghallab, M.; and Laruelle, H. 1994. Representation and Control in IxTeT, A Temporal Planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)*. Menlo Park, Calif.: AAAI Press.

Johnston, M. D. 1994. Spike: Intelligent Scheduling of Hubble Space Telescope Observations, 391–422. In *Intelligent Scheduling,* ed. Mark Fox and M. Zwebem. San Francisco: Morgan Kaufman, Publishers.

Jónsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*,177–186. Berlin: Springer-Verlag.

Koehler, J.; Nebel, B.; Hoffmann, J.; and Dimopoulus, Y. 1997. Extending Planning Graphs to an ADL Subset. In *Proceedings of the Fourth European Conference on Planning*. Berlin: Springer-Verlag.

Kvarnstrom, J.; and Doherty, P. 2001. TALplanner: A Temporal Logic based Forward Chaining Planner. *Annals of Mathematics and Artificial Intelligence* 30(1–4): 119–169.

Long, D., and Fox, M. 2001. Encoding Temporal Domains and Validating Temporal Plans. Paper presented at the Twentieth Workshop of the UK Planning and Scheduling Special Interest Group (PlanSig 2001), University of Edinburgh, Edinburgh, Scotland, 13–14 December.

Mieri, I. 1996. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. *Artificial Intelligence Journal* 87 (1–2): 343–385.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling,* ed. M. Fox and M. Zweben. San Francisco: Morgan Kaufman, Publishers.

Muscettola, N.; and Smith, B. 1997. On-Board Planning for New Millennium Deep Space One Autonomy. In *Proceedings of the 1997 IEEE Aerospace Conference*. Piscataway, NJ: The Institute of Electrical and Electronics Engineers.

Myers, K. L., and Berry, P. M. 1999. At the Boundary of Workflow and AI. In *Agent-Based Systems in the Business Context: Papers from the AAAI Workshop*, ed. Brian Drabble and Peter Jarvis. AAAI Tech. Report WS-99-02. Menlo Park, CA: American Association for Artificial Intelligence.

Pednault, E. 1989. ADL: Exploring the Middle Ground between Strips and the Situation Calculus. In *Proceedings of the First Conference on Principles of Knowledge Representation and Reasoning,* 324–332. San Francisco: Morgan Kaufmann, Publishers.

R-Moreno, M. D.; Borrajo, D.; and Meziat, D. 2002. An AI Tool for Planning Satellite Nominal Operations. Paper presented at the Third International NASA Workshop on AI Planning and Scheduling for Autonomy in Space Applications, Manchester, U.K., 6 July.

R-Moreno, M. D.; and Kearney, P. 2002. Integrating AI Planning with Workflow Management System. *International Journal of Knowledge-Based Systems* (Elsevier) 15(8): 285–291.

R-Moreno, M. D.; Oddi, A.; Borrajo, D.; Cesta, A.; and Meziat, D. 2004. IPSS: A Hybrid Reasoner for Planning and Scheduling. In *Proceedings of the Sixteenth European Conference on Artificial Intelligence (ECAI–04)*. Berlin: Springer-Verlag.

Rabideau, G.; Chien, S.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F.; Barrett, T.; Stebbins, G.; and Tran, D. 2000. ASPEN–Automating Space Mission Operations Using Automated Planning and Scheduling. Paper Presented at the Sixth International Symposium on Space Mission Operations and Ground Data Systems (SpaceOps 2000), Toulouse, France, 19–23 June.

Rabideau, G.; Knight, R.; Chien, S.; Fukunaga, A.; and Govindjee, A. 1999. Iterative Repair Planning for Spacecraft Operations in the Aspen System. Paper presented at the Fifth International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS). Noordwijk, The Netherlands, 1–3 June.

Sherwood, R.; Engelhardt, B.; Rabideau, G.; Chien, S.; and Knight, R. 2000. *Aspen User's Guide Version 2.0*. Tech. Rep. D-15482. Pasadena, CA: NASA Jet Propulsion Laboratory.

Smith, D.; Frank, J.; and Jónsson, A. 2000. Bridging the Gap between Planning and Scheduling. *Knowledge Engineering Review* 15(1): 61–94.

Smith, S.; and Cheng, C. 1993. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Srivastava, B.; Kambhampati, R.; and Do, M. B. 2001. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence Journal* 131(1–2): 73–134.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-Plan2: An Open Architecture for Command, Planning, and Control. In *Intelligent Scheduling,* ed. Mark Fox and M. Zwebem. San Francisco: Morgan Kaufman, Publishers.

Tate, A.; and Whiter, A. M. 1984. Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem. Paper Presented at the First Conference on the Applications of AI, Denver, Colorado. Technical Report AIAI-TR-4, Artificial Intelligence Applications Institute, School of Informatics, University of Edinburgh, Edinburgh, Scotland.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical Artificial Intelligence* 7(1): 81–120.

Vere, S. A. 1983. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 5(3): 246–267.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm*. San Francisco: Morgan Kaufman, Publishers

Willis, J.; Rabideau, G.; and Wilklow, C. 1999. The Citizen Explorer Scheduling System. In *Proceedings of the IEEE Aerospace Conference*. Piscataway, NJ: The Institute of Electrical and Electronics Engineers.

Maria Dolores Rodriguez-Moreno is a teacher assistant at the Universidad de Alcalá. She obtained her Ph.D. in computer science from the Universidad de Alcal in March of 2004 in the topic of integration of AI planning and scheduling. Her research interests are in the application of AI techniques to real problems such as workflow or satellite domains. She can be reached at mdolores@aut.uah.es.

Daniel Borrajo is a professor of computer science at the Universidad Carlos III de Madrid. He received his Ph.D. in computer science in 1990 and B.S. in computer science both at the Universidad Politécnica de Madrid. He has published more than 90 journal and conference papers mainly in the fields of problem-solving methods (planning and game playing) and machine learning. His e-mail address is dborrajo@ia.uc3m.es.

Daniel Meziat is a professor at the Universidad de Alcalá. His research has focused on instrument on-board satellites. His e-mail address is meziat@aut.uah.es.s