An AI Planning-based Tool for Scheduling Satellite Nominal Operations

María Dolores Rodríguez-Moreno [1], Daniel Borrajo [2], Daniel Meziat [1]

[1] Departamento de Automática. Universidad de Alcalá.
Ctra Madrid-Barcelona, Km. 33,6. 28871 Alcalá de Henares (Madrid), Spain.
{mdolores, meziat}@aut.uah.es

[2] Departamento de Informática. Universidad Carlos III de Madrid.
Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain.
dborrajo@ia.uc3m.es

Abstract

Satellite domains are becoming a fashionable area of research within the AI community due to the complexity of the problems that these domains need to solve. With the current US 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 the paper is to present CONSAT, a real application that allows to plan and schedule nominal operations to perform in four satellites along the year for a commercial Spanish satellites company: HISPASAT. We have used an AI domain independent planner for this task 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 operators variables 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.

1 Introduction

Complex real world tasks usually require the combination or integration of tools and techniques from two well-known fields, namely planning and scheduling [46]. An example of such domains are workflow applications that require generating sequences of activities that define a process in an organisation and the assignment of resources (human or material) to these activities [36, 39, 41]. Other examples are building aircrafts [19] or space missions control [4, 20, 28, 38, 44]. To solve problems in any of the domains mentioned above, we need to represent the information needed in order to find good solutions efficiently. 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 [21] is becoming a standard in the planning field for representing domains and problems. Although PDDL2.2 and other predecessor planning languages allow representing this type of real problems, in many cases some assumptions have to be made and in some cases the problem must be reduced.

Traditionally, these domains 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 [3]. They try to find plans to achieve a set of goals from an initial state, and are good at finding precedences among activities, but limited at resource or time reasoning.

On the other hand scheduling systems allocate available resources to known activities over time in order to produce schedules that respect temporal relations and resource capacity [11, 49]. They are good

at optimising and assigning time and resources to activities, but they require knowing ordered relations among the activities. They can optimise a set of objectives, such as minimising makespan, minimising work to be done, maximising resource allocation or minimising cycle time.

Depending on the problem complexity some domains allow a strict separation between planning and scheduling. A current simple approach to solve this problem is to use a scheduler at the output of the planner in order to assign resources and time to each activity [10]. But, in other cases, there is an indirect temporal and resource dependency with other states and goals that can not be taken into account if we separate both tasks [25]. The above approach is weak if the scheduler fails to find a solution: expensive and unsatisfactory solutions can be generated by the planner again if it does not receive any feedback from the scheduler.

One way of integrating both tasks within the same tool consists on adding representation and reasoning capabilities on resource and temporal information to any planner. This has been done in systems such as IxTeT [26], O-PLAN2 [16, 50], HSTS [34], RealPlan [48] or IPSS that we presented in [42]. In this paper, we wanted to explore the possibility of using a non-linear domain independent planner, PRODIGY [52], and study the possibility of using it directly for generating solutions to problems requiring both planning and scheduling. This planner does not have an explicit model of time representation nor a declarative way for specifying resource requirements or consumption. However, thanks to its capability of representing and handling continuous variables, coding functions to obtain variables values and the use of control rules to prune the search, we have successfully integrated planning and scheduling in a satellite control domain. This domain 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) for setting up nominal operations to perform in the satellites during the year.

The paper is structured as follows: section 2 describes the features of the planner. Sections 3 and 4 introduce the type of operations on satellites that have to be performed during the year and the knowledge represented in the planner. Section 5 presents the tool functionality and the modules that integrate it. Section 6 shows some experimental results. Finally, related work is described and conclusions are drawn.

2 Planning and Scheduling: PRODIGY

The planner we use to schedule the HISPASAT operations is PRODIGY [52], an integrated architecture that has been used in a wide variety of domains. The problem solver is a non-linear planner that uses a bidirectional means-ends analysis search procedure with full subgoal interleaving [7]. The planning process starts from the goals and adds operators to the plan until all goals are satisfied. Although it does not use a language that allows an explicit representation of resources or temporal information, it is able to handle some scheduling reasoning thanks to its capability to: represent infinite types (numeric variables) which are needed to represent information about time and resources; define functions that obtain variables values in preconditions of operators so that values can be constrained; and use control rules (heuristics) to prune the search, which allow to make the overall problem solving process efficient. Figure 1 shows the types of knowledge needed by the planner.

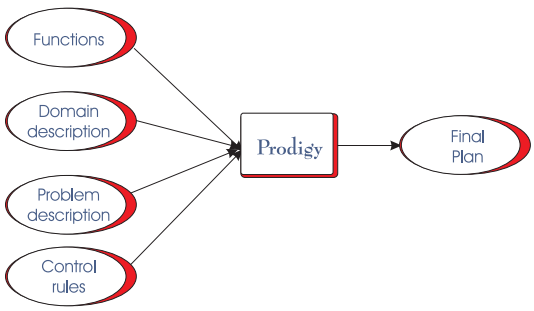

Figure 1: The inputs and output of the PRODIGY planner.

The reasons to choose this particular planner are manyfold. Among them, we can highlight defini-

tion and handling of quality metrics, explicit definition of control rules, flexibility to define new search behaviours 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 [7], is based on an augmentation of the STRIPS representation originally proposed by Fikes and Nilsson [23]. 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 pre-conditions (conditions that must be true to allow the action execution), and post-conditions 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. Since this representation is quite restrictive, it has been extended to allow disjunctive preconditions, conditional effects and universally-quantified preconditions and effects [7] resulting in a similar language to ADL [37]. Figure 2 shows an example of an operator in the PDL4.0 syntax in the HISPASAT domain. The operator represents the action in charge of the satellite sensor position maintenance. It is performed during the spring and autumn equinoxes. The symbols within < > are variables that are instantiated during problem solving. This operator has two preconditions: *(equinox-spring <d>)* and *(no-manoeuvre <s> <t> <d0>)* and just one add effect *(ires-transitioned <s>)*. As the variables <d> and <d0> are numbers, we need to use the *gen-from-pred* PRODIGY function to constraint the values that the numeric variable DATE can have. It generates a list of values to be possible bindings for the corresponding variable by using the information of the current state referred to the no-manoeuvre and equinox-spring literals. That is, this function permits encoding the functions d0= no-manoeuvre(s,t) and d = equinox-spring(d0).

```
(OPERATOR IRES-TRANSITION
   (preconds
     ((<s> SATELLITE)
       (<t> TIMES)
       (<d0> (and DATE (gen-from-pred (no-manoeuvre <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: Example of a PRODIGY operator in the HISPASAT domain.

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 has also in mind the temporal constraints among the operators. The obtained plan does not consider any optimisation 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 in [6].

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 have followed a similar approach, though using a less declarative definition of control knowledge expressed in a form of temporal logic, such as TALplanner [31]. There are three types of rules: selection, preference or rejection. One can use them to choose an operator, a binding, a goal, or deciding whether to apply an operator or continue sub-goaling. Figure 3 shows an example of a control rule to select a binding. From all batteries that can be used for reconditioning, this control rule selects the battery that is currently unloaded.

```
(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: Example of a control rule that decides what battery to choose.

3 Satellites maintenance operations

This section provides an overview of the HISPASAT ground scheduling operations for its four satellites. HISPASAT is a Spanish multi-mission 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 defense purposes. It is the first European satellites system to offer transatlantic capacity that allows 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, actually four, 1A, 1B, 1C and 1D, and two in the future, 1E and Amazonas, a program that generates and validates the schedule of operations for engineering support was needed.

In the HISPASAT domain, an important role in the schedule of the rest of operations is played by a special type of operation, the manoeuvre operations. The manoeuvre operations are in charge of the correct satellite positioning into orbit. They must follow a rigid time table: every two weeks (on Monday or Tuesday depending if it is summer or winter) in a specific hour given by an external software tool. These operations can only be moved ahead if any interference (by the sun or by the moon) occurs. Interferences cause wrong measures in the satellite orbit so the sensors affected by them must be masked using the adequate operations. Once these operations are scheduled, the operations related to the use of tanks or batteries are set: always in weeks without manoeuvres. Finally there is a small set of operations that only depend on the last time they were performed. In these cases, just the data of the last operation of their same type is needed.

The ways we have used for eliciting the knowledge that we describe in more detail below, have been: the standard written sources, open interviews, structured interviews, questionnaires, the documentation generated for each satellite in previous years and the satellite operation manuals.

3.1 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 structured in a hierarchy. A special kind of type, infinite types, allow to represent continuous valued variables, 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 given that under our domain formalisation only a finite number of values for them are considered. Finally, DATE is represented as an infinite type. Figure 4 shows one of the hundred operators that have been implemented in the HISPASAT domain. In particular it is the South-Manoeuvre operator, in charge of computing the date at which this operation can be performed having in mind that any moon blinding (represented by the *moon-blinding* predicate) cannot interfere with the expected South-Manoeuvre date within three hours. If this occurs, the operation will be moved 24 hours ahead.

As mentioned before, the PRODIGY function *gen-from-pred* generates a list of values to be possible bindings for the corresponding variable by using the information of the current state. In this particular

Table 1: South Manoeuvre task. This is one of the South-Manoeuvres that will be performed during year 2004.

Name	South-Manoeuvre
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 manoeuvre 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

```
(OPERATOR SOUTH-MANOEUVRE
    (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-Manoeuvre <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-manoeuvre <s> <t> <d1>))
    (effects
        ((add (south-m <s> <t>))
         (add (south-man <s> <t> <start-time>)))))
```

Figure 4: Operator corresponding to the South-Manoeuvre task of Table 1. If there is a moon blinding within three hours from the expected manoeuvre with intensity over 40, the manoeuvre has to be moved ahead 24 hours. The end time of the task is calculated by the *calculate-end-time* function.

case we use it to generate all the values that the numeric variable DATE can have. DATE represents seconds since 1900 in GMT (we have used Common Lisp as the programming language for functions given that 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 [34, 50, 53].

The rest of 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 re-used in any such domain as it will be described later on. As an example of domain dependent functions, *Is-South-Manoeuvre* generates the date of the manoeuvre (if there is any) that overlaps within three-hours any moon blinding. If the blinding intensity is over 40%, the manoeuvre 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 manoeuvre, 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.

With respect to the pre-conditions of this operator, it has only one: *(south-manoeuvre <s> <t> <d1>)*, that is, the date at which the South-Manoeuvre is expected to be performed (as shown in Figure 6 this is part of the initial state).

The operator has two effects that belong to the add list; the predicates *south-m* that is the goal that we want to achieve, and *south-man* that adds to the state the date at which the South-Manoeuvre must be performed. In case any interference occurs within three hours, the value of the <start-time> variable matches the value of the expected manoeuvre, <d1>, moved ahead 24 hours.

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

3.1.1 Operations which are driven by external events and shall start or be completed at a fixed time

Some operations depend on external events such as Moon Blindings, Sun Blindings or Eclipses of the Sun by the Earth or by the Moon. These events constraint when other operations can be performed. Given that these are hard constraints, they are represented as pre-conditions of the operators. Some external (i.e. eclipses and blindings) and seasonal (i.e. 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 their 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 are in charge of setting masks onto the upper or lower position detectors depending 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 in spring and autumn equinoxes, switching on or off level heaters to compensate seasonal evolution. 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 the operators effects.

3.1.2 Operations that are part of a given strategy and should be performed at specific dates

Other operations have more flexibility in the date at which they can be performed. This is the case of manoeuvres, Localisation Campaigns or Batteries Reconditioning. Given that they are not hard constraints, we have coded functions in the pre-conditions to guide the planner for preferring some dates over others having in mind the restrictions imposed in some pre-conditions. This decision restricts the number of possible solutions that the planner will provide, but 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 having in mind the date when a South-Manoeuvre was performed. These operations are in charge of the correct satellite orbit positioning, and they are performed every two weeks at a specific time.[1] In the initial state, we define the satellite affected by the secular drift, and the date and time of the year during which the Manoeuvre is going to be performed. Table 1 shows the features of a South-Manoeuvre. Given that the manoeuvres are forbidden during moon or sun blindings, they have to be moved ahead 24 or 48 hours (in case two moon blindings appear in following days) from the secular drift time. The two manoeuvres (West and East-Manoeuvre) that follow the first one must also be moved ahead the same number of hours. The rest of operations in this category start/finish some hours before/after the start/end of the South-Manoeuvre was performed. Some operations cannot be performed if a Manoeuvre has been scheduled during that week, while others must be performed during/after/before Manoeuvres. Others just have to be performed N days/weeks/months

[1] 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.

since the last time they were performed. Therefore, we had to represent different types of temporal relations among operators.

In order to represent this knowledge, we needed a problem solver able to express the fact that if a South-Manoeuvre is performed during one day of a week (having in mind Moon and Sun Blindings), other operations, such as BOOST HEATING, can not be performed during the same week, but the following week, or the CONF-ADCS operation that must be performed 9 hours after the East-Manoeuvre. This kind of reasoning is hard to represent in the operator preconditions and difficult to solve by most current planners. PRODIGY can solve them thanks to the coded functions that restrict the value of the BOOST HEATING start time variable to the week after the South-Manoeuvre, as we will describe later in the paper.

3.1.3 Operations that should be performed within a long interval

A significant flexibility exists for scheduling some operations, as it is the case of the Steerable Antenna Maintenance. This operation is performed four times a year. So, in order to schedule it, we need to know the last date at which the operation was performed. The operations that belong to 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 these operations, we only need to represent in the initial state the last time the operation was performed in each satellite during the previous year. As with respect to the goals, we have to introduce the number of executions of each operator during the year (in the above example, four times).

3.2 Control Rules

In the HISPASAT domain, we identified as resources the tanks and the batteries. Control rules can help to assign a given resource to an operation, for example a tank. Each satellite has four tanks. Any of them could be chosen for the swapping operation, but there is a recommendation from the satellites builder company 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 operator TANK-SWAPPING, 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.

```
(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.

We could have alternatively coded a function to choose the correct tank for the TANK-SWAPPING operation in the precondition of the operator. However, coding this type of preferences 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.

3.3 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 during which it will take place (3282941400 = 1/13/2004 01:10:00), the direction (south) and the intensity (45% represented by p-45).

Finally, the moon blinding chronological appearance, that is, the first time it appears is represented by t1, the second time t2, etc. With respect to the goals, we need to perform a fixed number of manoeuvres during the year, which is represented in the goals section of the problem.

```
(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-manoeuvre 1B t1 3257107800)
      (south-manoeuvre 1B t2 3258317400) ...))
(goal
   (and
      (south-m 1B t1) (south-m 1B t2)...  ))
```

Figure 6: Some goals and initial conditions for the 1B satellite. For instance, in the goals it is required that the South-Manoeuvre should 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 in case there are two or more numerical variables in the same predicate, then PDDL2.2 cannot handle it directly.

3.4 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 mentioned on section 3, manoeuvres are the main operations. Once they are scheduled, the rest can also be scheduled. But they are scheduled having in mind the atmospheric conditions (blindings and eclipses). These conditions could be modelled as binary resources with a fixed start and end time. Any scheduler will try to schedule the manoeuvres when *these resources* are not being used. However, this will not be a feasible solution for HISPASAT as manoeuvres can only be moved 24 or 48 hours ahead and, in the last case, some operations can be omitted or scheduled/planned in a different way depending on the date that the manoeuvre 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.

4 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 [2]. First, we describe the time aspect of the scheduling. Then, we describe some issues about resource usage in this domain.

4.1 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. There is no provision for allowing parallel actions. However, the functions that can be called within the preconditions of the operators when assigning values to variables

allow adding constraints among and within operators. Using them, PRODIGY can handle the seven Allen primitive relations between temporal intervals [2] and some quantitative relations [18, 33]. [2]

PDDL2.1 is also a discrete model of time, that considers time representation at level 3 by means of temporal conditions and effects in 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 allows concurrent behaviour if it is avoided that another action accesses a variable at the exact point at which it is updated by another action. So conflicts over variables can only occur at the start and end points of actions. In the preconditions, the propositions can be asserted at the start of the interval (the point at which the action is applied), at the end of the interval (the point at which 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 to 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 [32]. 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 in Partial Order Plans it must be had in mind as shown in [15]. On the other hand, PDDL presents more restrictions than PDL4.0 to represent the Allen primitives, because it 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 time point represents its start time and the second one 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.

In order 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, the values returned can be as many as one needs, so, at 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). On the contrary, in the CSP representation constraints are established much more easily.

The five categories are:

- 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 time 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

[2] Due to the fact that 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.

Table 2: PRODIGY domain independent coded funtions to handle time.

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 ELAPSED^1 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.

```
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.

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 operator B. On the contrary, PRODIGY can wait until OperatorA is applied to compute when to start execution of OperatorB.

```
(: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 DUR^0)
 :condition (at start (and (>= (+ (end-A) ELAPSED^1) (start-B))
                           (<= (+ (end-A) ELAPSED^2) (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.

- 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 the case we would like to represent the *equals* relationship we should also constraint 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.

```
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>
                        ELAPSED^1 TIME^1 ELAPSED^2 TIME^2))
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.

In PDDL2.2 we can impose the same start time in both operations and the duration of the OperatorB can be greater than the end time of the OperatorA minus ELAPSED^2 and less than the end time of the 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 belongs

```
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.

to this category. The case OperatorA *equals* OperatorB could have also been in this category (where a=b=0). In 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 that the end time of both activities is equal. In case of OperatorA *equals* OperatorB primitive the value of the duration of OperatorB should be equal to the duration of OperatorA. To impose the ending of both operators at the same time, this 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.

```
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.

- 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.

```
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>
                          ELAPSED[1] TIME[1] ELAPSED[2] TIME[2]))
           (<end-time> (add-time <start-time> DUR[0] TIME[0]))
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.

- 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.

4.2 Enriching Time Constraints

The relations between operators described in the previous section, are defined within a time window and considering 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:

```
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>
                          ELAPSED^1 TIME^1 ELAPSED^2 TIME^2))
           (<end-time> (del-time-in-interval <d1>
                        ELAPSED^3 TIME^3 ELAPSED^4 TIME^4))
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.

- *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 being used for any of the temporal relationships between operators and a global counter to compute the last value used.

Given this solution requires some modelling 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.

4.3 Representation of Resources Constraints

A resource is a source of supply or support or an available means. There are basically three types of resources [45], although the name given to each type varies from one author to another:

- Type 1: it 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 [45] a similar type is the *concurrency* resource that must be

```
                    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.

made available to the activity before they are reserved. An example would be a tele-communications downlink pass. The telecommunications station must be available before the spacecraft could initiate a downlink.

- Type 2: it 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.
- 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 (can be replenished) and fuel (it cannot 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 [48] show.

For example, if tank T1 of a satellite has 30 litres of fuel in a given instant, we could represent it as *(has-fuel T1 30)*. Then, in the operators, we can restrict the set of values that can be assigned to the variable that represents the resource, consume it, or refuel it 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 (i.e. *compute-consumed-fuel*) that calculates the fuel consumed in each manoeuvre 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>)* that must be greater that 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-manoeuvre <sat>)* and *(available-fuel <sat> <new-fuel>)* to the state, i.e. it asserts them as true, and deletes *(available-fuel <sat> <available>)* 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 resources consumption is the same, as Figure 21 shows.

```
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-manoeuvre <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 Manoeuvre-S
 :parameters (?sat - satellite)
 :precondition (> (position ?sat) 4)
 :effect (at end (performed-manoeuvre ?sat))
         (at end (decrease (available-fuel ?sat)
                           (/ (available-fuel ?sat) (position ?sat)))))
```

Figure 21: PDDL2.2 representation of Figure 20 operator.

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 its 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 IPC'02 [27]. 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).

```
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.

In CSP scheduling, there are many algorithms and heuristics to easily handle from binary to multicapacity resources as the ones proposed in [8, 9, 47].

```
(: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.

5 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 [40]. The CONSAT[3] tool consists of the following subsystems that 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 Susbsystem:** 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 allows to convert the solution to a specific format document type, compare two different solutions, or generate 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.

5.1 The User Subsystem

The User Subsystem is in charge of data introduction. The control engineers are the ones that 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 is difficult to provide by a user directly such as the time in seconds that most operators and literals need as arguments. Therefore, the user subsystem allows the user to specify data in their current usual way and the subsystem automatically translates it into the internal model.

This subsystem provides the following more specific functionality:

- In the case of external events and South-Manoeuvres, 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 at which the South-Manoeuvre can be performed every day of the year. These software tools generate the data in a set of ASCII files. The subsystem allows importing these files and detecting the correct file format.

[3]It stands for SATellite CONtrol in Spanish.

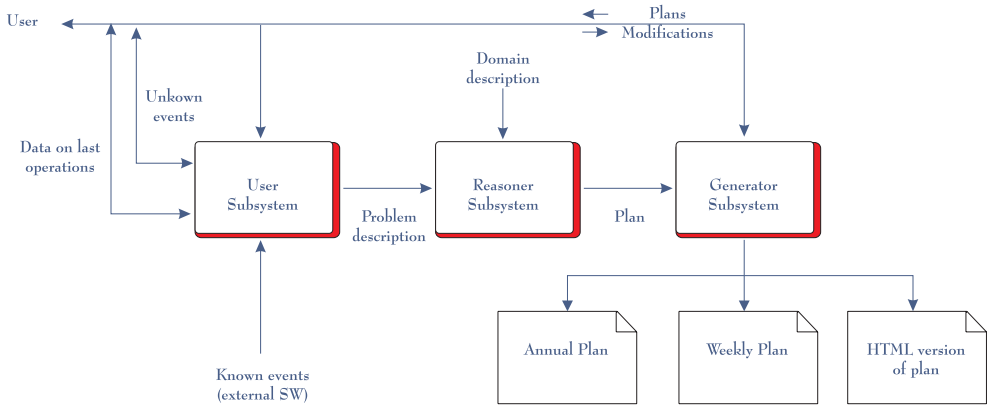

Figure 24: Architecture of the planning tool.

- 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 (section 3.1.3), they require to know the data at which they were performed the last time, so the engineers must find in the last year's document when these operations were performed. If the tool has been previously used, this data is available to the user subsystem and automatically re-used.
- Finally, it assists the user in validating the introduced data. This allows to avoid the possible user inconsistencies, such as trying to swap twice the same tank in the same period of the year or perform an operation in an illegal period.[4] Figure 25 shows one of the interface windows (written in Spanish due to engineers requirements) of the User Subsystem. It shows in the left part of the window an ordered list of operations that require user inputs. In the right hand 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.

5.2 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 in section 2. It is in charge of the plan generation, with temporal and resource reasoning. It generates a problem file with all the information introduced by the user. This file joined to the already defined domain and control knowledge files allows to run the planner. The planner output is saved into another ASCII file that will be manipulated to generate the input to the Generator Subsystem.

5.3 The Generator Subsystem

Currently, once the engineers know every external event and have represented them in a document, the laborious task of scheduling every operation starts. They generate two types of documents:

- A document which provides an overview of all the operations performed each day of the year; and
- A document that represents in more detail each operation duration, type of manoeuvre, the resources, if any, affected by the operation, such as batteries or tanks, etc.

[4] A legal period in TANK SWAPPING is a date around equinox, for instance.

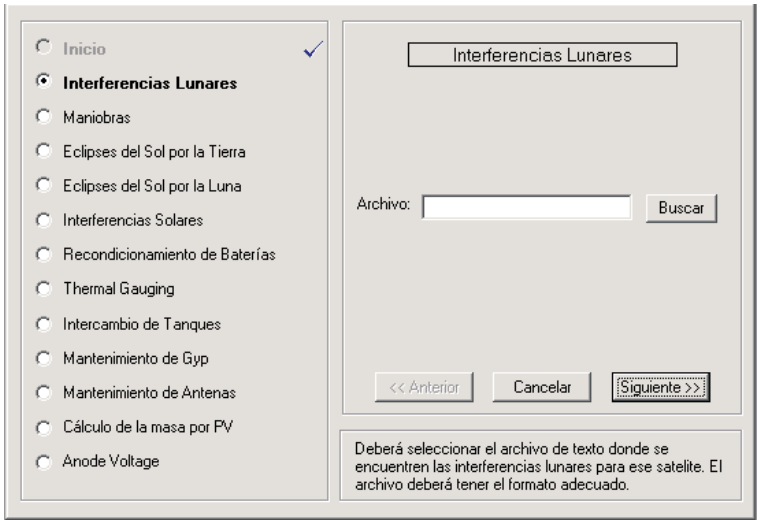

Figure 25: One of the windows of the User Subsystem interface.

The weekly representation is generated every week having in mind 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 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 backup centers.

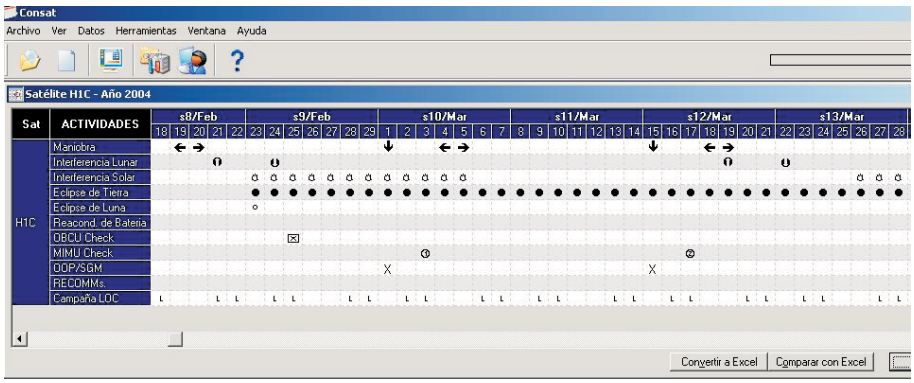

Figure 26: The Generator Subsystem interface: annual representation.

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

6 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 in a Pentium III 800 MHz and 256Mb under Windows. The number of operators in the HISPASAT domain is of 100, the number of control rules is ten and a total of

Table 3: Results obtained to achieve a plan for each satellite.

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

50 coded functions, 25% of them domain independent.

In order to compare it with humans, the time devoted to prepare the annual representation for the three satellites is 40 hours a year by one person: basically 35 hours to elaborate it and five hours for verification. For the weekly representation, they 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 them are valid. We analyzed differences with the real ones provided by human experts, and the results were that the plans generated by the tool agree in at least 90% of the times with the plans generated by the engineer team. The main differences fall on last time operation changes due to the satellites needs, or coincidence of some operations with holidays which force to move some operations to other dates. This last discrepancy will be considered in future versions of the software. CONSAT is actually under the validation phase.

7 Related Work

Several planning systems have addressed applications of planning with resources and time considerations. The planner NONLIN+ [51] maintains information about the duration of the activities and represents limited resources. SIPE [54] allows the declaration of the use of resources in the operators. DEVISER [53] could also handle consumable resources. These planners have been designed with the purpose of explicitly handling some types of resources. In the IPP planner [30], based on Graphplan [5], 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 [48], also based on Graphplan, decouples the logical reasoning from resources reasoning thanks to the CSP formulation that allows helping the planner in the resources allocation. This is the main difference with PRODIGY that has been designed as a general planning and learning system rather than 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 sections 3 and 4).

Other current approaches have integrated planning and scheduling in the same system as HSTS [34] by instantiating state-variables into a temporal database. These state-variables are very different to predicate logic formulae as used by PRODIGY, although it is a merely convenient shorthand for logic, leading to no loss in expressivity. Also the O-PLAN2 [50] 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 [26]. 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 multi-valued domain attributes temporally qualified into instantaneous events and persistent assertions and a set of resource attributes. SPIKE [29] 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. MAPGEN [1] 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 is called EUROPA [24]. It is an evolution of the Deep Space One planner [35]. 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.

They all 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 allows introducing time management in an intuitive way almost equivalent to the expressive power of other tools.

Also ASPEN [44] uses an AI planner with a richer representation in activities that easily allows the introduction of start times, end times, duration and use of one or more resources. Some algorithms that this planner uses to solve the problem belong basically to the scheduling area (i.e., the schedule iterative repair algorithm). It is part of CASPER (Continuous Activity Scheduling Planning Execution and Replanning) [43] that allows working in dynamic environments as most of NASA projects [12, 13, 14, 22, 55] require. On the other hand, our planner uses specific algorithms of the planning area that can handle a subset of the scheduling problem that is enough for solving tasks in the HISPASAT domain. We do not need replanning, operations monitoring or execution as the operations are not as critical as on-board missions and the planning execution time is not a high priority issue. 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) [17]. The way they face the problem differs from ours in several aspects. First, GPSS takes as inputs the set of precedence relation between tasks. It allows four types of temporal constraints with intervals between two tasks: activity-A *finishes* before 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 modelled 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 it is not enough the four precedence relations between tasks as we need other Allen primitives 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, consider 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 [48], 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 any other classical planner, an operator consists of preconditions, and effects. Actions have well defined their start and end times and resource assignments for each activity. The obtained plan does not consider any optimisation 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 [6].

8 Conclusions and Future Work

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

The tool not only saves a lot of time to the user due to its capability of importing files of any type, representing the results in a table that allows easily to modify them by just drag and drop, generate more than one solution, see the differences between any two solutions, and generate the result in their internal format or in HTML. But there is also a high level description language to specify satellite problems (domain

dependent knowledge) and planning and scheduling problems (domain independent knowledge) that easily allows the tool maintenance, adding the correspondent operations when new satellites are added.

We want to explore in the future the possibility of adding more satellites allowing CONSAT to define/delete new operations, to adapting it to new satellites as MAPGEN [1] is adapted to different missions. This would allow us the study of the scalability of the approach for dealing within the planner with planning, temporal and resource reasoning. It would be also interesting to integrate a monitoring module to execute the scheduled operations obtained by the planner and to re-plan in case of changes in the plan. Finally, we would like to explore its generality by using the domain temporal independent knowledge for coding other domains that require planning and scheduling reasoning.

Acknowledgements

We want to thank all the HISPASAT Engineering Team for their help and collaboration shown during the developing time 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.

References

[1] Ai-Chang, M., Bresina, J., Charesty, L., Hsu, J., Jónsson, A., Kanefsky, B., Maldaguey, P., Morris, P., and Rajan, K. Yglesias, J. MAPGEN Planner: Mixed-Initiative Activity Planning for the Mars Exploration Rover Mission. In *Procs. of the 7th International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)* (2003).

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

[3] Allen, J. F., Hendler, B., and Tate, A. *Readings in Planning.* Morgan Kaufman, 1990.

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

[5] Blum, A., and Furst, M. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence 90* (1997), 281–300.

[6] Borrajo, D., Vegas, S., and Veloso, M. Quality-Based Learning for Planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources.* (2001).

[7] 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. PRODIGY4.0: The Manual and Tutorial. Tech. Rep. CMU-CS-92-150, Department of Computer Science, 1992.

[8] Cesta, A., and Oddi, A. Algorithms for Dynamic Management of Temporal Constraint Networks. Tech. rep., Italian National Research Council (ISTC-CNR), 2002.

[9] Cesta, A., Oddi, A., and Smith, S. F. Greedy Algorithms for the Multi-Capacitated Metric Scheduling Problem. In *Proceedings 1999 European Conference on Planning* (1999).

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

[11] Cheng, C. C., and Smith, S. F. Applying Constraint Satisfaction Techniques to Job Shop Scheduling. Tech. Rep. CMU-RI-TR-95-03, Robotics Institute, 1995.

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

[13] Chien, S., Rabideau, G., Willis, J., and Mann, T. radarsat-mamm Automated Mission Planner. *AI Magazine 23*, 2 (2002), 25–36.

[14] 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. Autonomous Science on the EO-1 Mission. In *Procs. of the 7th International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)* (2003).

[15] Coddington, A. A Continuous Planning Framework with Durative Actions. In *9th International Symposium on Temporal Representation and Reasoning, TIME-2002. Manchester, UK. IEEE Computer Society, 2002.* (2002), pp. 108–118.

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

[17] Deale, M., Yvanovich, M., Schnitzius, D., Kautz, D., Carpenter, M., Zweben, M., Davis, G., and Daun, B. *Intelligent Scheduling.* Morgan Kaufman, 1994, ch. The Space Shuttle Ground Processing Scheduling System., pp. 423–449.

[18] Dechter, R., Meiri, I., and Pearl, J. Temporal Constraint Networks. *Artificial Intelligence 49* (1991), 61–95.

[19] Drabble, B., McVey, C., and Clements, D. Agile Aircraft Manufacturing and Assembly. In *Proceedings of the Symposium on Planning, Scheduling, and Control for Aerospace at the World Congress on Automation (WAC-2000)* (2000).

[20] Dungan, J., Frank, J., Jónsson, A., Morris, R., and Smith, D. Planning and Scheduling for fleets of Earth Observing Satellites. In *Procs. of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space i-SAIRAS.* (2001).

[21] Edelkamp, S., and Hoffmann, J. pddl2.2: The Language for the Classical Part of the 4th International Planning Competition. Tech. Rep. Technical Report No. 195, 2004.

[22] Estlin, T., Rabideau, G., Mutz, D., and Chien, S. Using Continuous Planning Techniques to Coordinate Multiple Rovers. In *Procs. of the IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World.* (1999).

[23] Fikes, R., and Nilsson, N. strips: A new Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence 2* (1971), 189–208.

[24] Frank, J., Jónsson, A., and Morris, P. On Reformulating Planning as Dynamic Constraint Satisfaction. In *Procs. of the Symposium on Abstraction, Reformulation and Aproximation (SARA).* (2000).

[25] Garrido, A., and Barber, F. Integrating Planning and Scheduling. *Applied Artificial Intelligence* (2001).

[26] Ghallab, M., and Laruelle, H. Representation and Control in IxTeT, a Temporal Planner. In *Procs. of the Second International Conference on AI Planning Systems (AIPS-94)* (1994).

[27] IPC-2002. International Planning Competition. In *http://www.dur.ac.uk/d.p.long/IPC* (2002).

[28] Jónsson, A., Morris, P., Muscettola, N., Rajan, K., and Smith, B. Planning in Interplanetary Space: Theory and Practice. In *International Procs. of the Conference on Artificial Intelligence Planning Systems* (2000), pp. 177–186.

[29] Johnston, M. D. *Intelligent Scheduling.* Morgan Kaufman, 1994, ch. spike: Intelligent Scheduling of Hubble Space Telescope Observations, pp. 391–422.

[30] Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulus, Y. Extending Planning Graphs to an ADL subset. *Procs. of the ECP97* (1997).

[31] Kvarnstrom, J., and Doherty, P. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Articial Intelligence 30* (2001), 119–169.

[32] Long, D., and Fox, M. Encoding Temporal Domains and Validating Temporal Plans. In *Procs. of the 20th Workshop of the UK Planning and Scheduling Special Interest Group. PLANSIG2001. The Open University, UK* (2001).

[33] Mieri, I. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. *Artificial Intelligence, 87* (1996), 343–385.

[34] Muscettola, N. hsts: Integrating Planning and Scheduling. In *Intelligent Scheduling*, M. Zweben and M. Fox, Eds. Morgan Kaufman, 1994.

[35] Muscettola, N., and Smith, B. On-Board Planning for New Millenium Deep Space One Autonomy. In *Procs. of IEEE Aerospace Conference, Snowmass, CO.* (1997).

[36] Myers, K. L., , and Berry, P. M. At the Boundary of Workflow and AI. In *AAAI-99 Workshop on agent-Based Systems in The Business Context. Brian Drabble and Peter Jarvis Cochairs.* (1999).

[37] Pednault, E. adl: Exploring the Middle Ground between strips and the Situation Calculus. In *Procs. of the 1st Conference on Principles of Knowledge Representation and Reasoning* (1989), pp. 324–332.

[38] PLANET. Aerospace TCU Road Map. In *http://pst.ip.rm.cnr.it/en/events/planet/.*

[39] PLANET. Workflow Management TCU Road Map. In *http://scalab.uc3m.es/ dborrajo/planet/wm-tcu/.*

[40] R-Moreno, M. D., Borrajo, D., and Meziat, D. An AI Tool for Planning Satellite Nominal Operations. In *Workshop on AI Planning and Scheduling for Autonomy in Space Applications, Manchester (U.K.)* (2002).

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

[42] R-Moreno, M. D., Oddi, A., Borrajo, D., Cesta, A., and Meziat, D. ipss: Integrating Hybrid Reasoners for Planning and Scheduling. In *Procs. of the 16th European Conference on Artificial Intelligence, ECAI04* (2004).

[43] 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. aspen - Automating Space Mission Operations using Automated Planning and Scheduling. In *SpaceOps 2000, Toulouse, France* (2000).

[44] Rabideau, G., Knight, R., Chien, S., Fukunaga, A., and Govindjee, A. Iterative Repair Planning for Spacecraft Operations in the aspen System. In *Procs. of the 5th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS), Noordwijk, The Netherlands.* (1999).

[45] Sherwood, R., Engelhardt, B., Rabideau, G., Chien, S., and Knight, R. aspen User's Guide. Version 2.0. Tech. Rep. D-15482, Jet Propulsion Laboratory, 2000.

[46] Smith, D., Frank, J., and Jónsson, A. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review 15*, 1 (2000).

[47] Smith, S., and Cheng, C. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Procs. of the 11th National Conference on AI (AAAI-93)* (1993).

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

[49] Tate, A., Drabble, B., and R., K. *Intelligent Scheduling.* Morgan Kaufman, 1994.

[50] Tate, A., Drabble, B., and R., K. *O-Plan2: An Open Architecture for Command, Planning, and Control.* Morgan Kaufman, 1994.

[51] Tate, A., and Whiter, A. M. Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem. In *First Conference on the Applications of AI* (1984), Denver, Colorado, USA.

[52] Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., and Blythe, J. Integrating Planning and Learning: The prodigy Architecture. *Journal of Experimental and Theoretical AI 7* (1995), 81–120.

[53] Vere, S. A. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence pami-5* (1883).

[54] Wilkins, D. E. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufman, 1988.

[55] Willis, J., Rabideau, G., and Wilklow, C. The Citizen Explorer Scheduling System. In *Procs. of the IEEE Aerospace Conference.* (1999).