APSI Case#1: Pre-planning Science Operations in Mars Express

Amedeo Cesta, Simone Fratini, Angelo Oddi, and Federico Pecora

ISTC-CNR, National Research Council of Italy
Institute for Cognitive Science and Technology, Rome, Italy
 ⟨name⟩.⟨surname⟩@istc.cnr.it

Abstract

This paper gives a preliminary overview on an on-going effort to build a planning system to support ESA mission planners in Long Term Planning for Mars Express. The output of the tool is a pre-optimized skeleton plan of the communication windows and spacecraft maintenance slots, that is used as a support in the dialogue between science and operation team in early pre-planning of activities. To capture details of the domain we have specialized the OMPS planning architecture to obtain a rapid prototyping tool that can support the knowledge engineering effort in this domain.

1. Introduction

This paper describes our current experience within the Advance Planning and Scheduling Initiative (APSI), a research project funded by ESA-ESOC and started in November 2006. The goal of the APSI is twofold. On one hand, the initiative is aimed at creating a software framework to improve the cost-effectiveness and flexibility of mission planning support tool development. On the other, the APSI strives to bridge the gap between advanced Artificial Intelligence (AI) planning and scheduling technology and the world of space mission planning. The final output of the project will be a general software framework for developing mission planning systems and three different case studies to demonstrate the validity of the proposed approach. Currently, we are developing the first case (called APSI Case#1), which has the goal of generating a pre-optimized skeleton plan for the Mars Express Long Term Planning (LTP) phase.

The Mars Express spacecraft features a fixed, body-mounted high-gain antenna for Earth communications, which introduces an essential incompatibility between simultaneous pointed scientific instrument observations and Earth data communications. On board there are nine payloads (including the Melacom radio receiver) which are still working and have taken a total of about 25,000 observations. Planning the activities of Mars Express is a complex process involving different stages and the interaction among several teams (Science Working Teams, Mission Analysis Team, Flight Dynamics and Flight Control Teams). Generally, we can identify three phases for planning: long, medium and short term planning, which respectively have durations of six months, one month and one week. Along these phases the planning activities of the satellite are incrementally refined to a granularity of one week. The Mars Express planning domain is characterized by several kinds of constraints, such as bounded on-board memory capacities, limited communication windows over the down-link channels, deadlines and ready times imposed by the payload requirements, as well as different sources of uncertainty — e.g., the amount of data generated at each scientific observation or the channel data rate. These constraints make the Mars Express program a challenging and interesting domain for research in automated problem solving.

In the next sections we describe how we have addressed the problem of automatically synthesizing Mars Express LTPs using the Open Multi-CSP Planning and Scheduling architecture (OMPS). This software framework evolves from our own previous experience [10] and from other approaches to planning based on timeline synthesis [6, 9, 11]. In developing Case#1, we apply a flexible and adaptable software development methodology inspired from the Agile software development methods and Extreme Programming [8, 7]. This paper describes how starting from the modeling capabilities of the OMPS framework, we have captured the relevant features of the Case#1 problem and then implemented a first version of a solving algorithm. Specifically, the component-based nature of OMPS is the key enabling feature which allows to achieve prototypical planning support tools for rapid knowledge elicitation and user evaluation cycles. As we show in this paper, the OMPS framework allows to reduce application development to (1) designing components, and (2) modeling the constraints which describe how components interact.

The paper is organized as it follows. First, we briefly introduce the OMPS general framework. Second, we describe our current prototyping experience, the Case#1. Next, we show how Case#1 is modeled for resolution with OMPS. Some conclusions end the paper.

2. OMPS General Software Framework

The OMPS architecture is a constraint-based integrated planning and scheduling architecture. Its most innovative feature is that it introduces the notion of domain components as a primitive entity for knowledge
Behaviors
State variable. Their associated consistency features are the following. Also on the representation of the behavior.
representations for this function can be used, depending on the details of its solving process, which have been developed independently.
In this section we briefly summarize OMPs architecture, although the enabling factor of its solving process is outside the scope of this paper. As we will see, the component-based nature of OMPs is the enabling factor for fast prototyping within the space mission planning context.

2.1 Components and Behaviors

In OMPs, a component is an entity that has a set of possible temporal evolutions over an interval of time $\mathcal{H}$. The component's evolutions over time are named behaviors. Behaviors are modeled as temporal functions over $\mathcal{H}$, and can be defined over continuous time or as stepwise constant functions of time. It is in general possible to provide different representations for a component's behaviors, depending on (1) the chosen temporal model (continuous vs. discrete, or time point based vs. interval based), (2) the nature of the function's range $D$ (finite vs. infinite, continuous vs. discrete, symbolic vs. numeric), and (3) the type of function which describes a behavior (general, piecewise linear, piecewise constant, impulsive and so on).

Not every function over a given temporal interval can be taken as a valid behavior for a component. The evolution of components in time is subject to "physical" constraints (or approximations thereof). We call consistent behaviors the ones that actually correspond to a possible evolution in time according to the real-world characteristics of the entity we are modeling. A component's consistent behaviors are defined by means of consistency features. In essence, a consistency feature is a function $f^\mathcal{C}$ which determines which behaviors adhere to physical attributes of the real-world entity modeled by the component.

It is in general possible to have many different representations of a component's consistency features: either explicit (e.g., tables or allowed bounds) or implicit (e.g., constraints, assertions, and so on). For instance, let us model a light bulb component. A light bulb's behaviors can take three values: "on", "off" and "burned". Supposing the light bulb cannot be fixed, a rule could state that any behavior that takes the value "burned" at a time $t$ is consistent if and only if such a value is taken also for any time $t' > t$. This is a declarative approach to describing the consistency feature $f^\mathcal{C}$. Different actual representations for this function can be used, depending also on the representation of the behavior.

A few more concrete examples of components and their associated consistency features are the following.

State variable. Behaviors: piecewise constant functions over a finite, discrete set of symbols which represent the values that can be taken by the state variable. Each behavior represents a different sequence of values taken by the component. Consistency Features: a set of sequence constraints, i.e., a set of rules that specify which transitions between allowed values are legal, and a set of lower and upper bounds on the duration of each allowed value. The model can be for instance represented as a timed automaton [2] (e.g., the three state variables in the figures on Subsection 3.1). Note that a distinguishing feature of a state variable is that not all the transitions between its values are allowed.

Resource (renewable). Behaviors: integer or real functions of time, piecewise, linear, exponential or even more complex, depending on the model you want to set up. Each behavior represents a different profile of resource consumption. Consistency Feature: minimum and maximum availability. Each behavior is consistent if it lies between the minimum and maximum availability during the entire planning interval. Note that a distinguishing feature of a resource is that there are bounds of availability.

The component-based nature of OMPs allows to accommodate pre-existing solving component into larger planning contexts. For instance, it is possible to incorporate the MEXAR2 application [4] as a component, the consistency property of which is not computed directly on the values taken by the behaviors, but as a function of those behaviors.¹

2.2 Component Decisions and Relations

Now that we have defined the concept of component as the fundamental building block of the OMPs architecture, the next step is to define how component behaviors can be altered (within the physical constraints imposed by consistency features).

We define a component decision as a pair $\langle \tau, \nu \rangle$, where $\tau$ is a given temporal element, and $\nu$ is a value. Specifically, $\tau$ can be: (a) a time instant (TI) $t$ representing a moment in time; (b) a time interval (TIN), a pair of TIs defining an interval $[t_s, t_e]$ such that $t_e > t_s$. The specific form of the value $\nu$ depends on the type of component on which the decision is defined. For instance, this can be an amount of resource usage for a resource component, or a disjunction of allowed values for a state variable.

Regardless of the type of component, the value of any component decision can contain parameters. In OMPs, parameters can be numeric or enumerations, and can be used to express complex values, such as "transmit bitrate" for a state variable which models a communications system. Further details on value parameters will be given in the following section.

Overall, a component decision is something that happens somewhere in time and modifies a component's behaviors as described by the value $\nu$. In OMPs, the consequences of these decisions are computed by the

¹Basically, it is computed as the difference between external uploads and the downloaded amount stated by the values taken by the behaviors [4].
components by means an update function $f^U$. This is a function which determines how the component’s behaviors change as a consequence of a given decision. In other words, a decision changes a component’s set of behaviors, and $f^U$ describes how. A decision could state for instance “keep all the behaviors that are equal to $d’$ in $t_1$” and another decision could state “all the behaviors must be equal to $d’’$ after $t_2$”. Given a decision on a component with a given set of behaviors, the update function computes the resulting set.

Let us instantiate the concept of decision for the two types of components we have introduced so far.

**State variable.** Temporal element: a TIN. Value: a subset of values that can be taken by the state variable (the range of its behaviors) in the given time frame. Update Function: this kind of decision for a state variable implies the choice of values in a given time interval. In this case the subset of values are meant as a disjunction of allowed values in the given time interval. Applying a decision on a set of behaviors entails that all behaviors that do not take any of the chosen values in the given interval are excluded from the set.

**Resource (renewable).** Temporal element: a TIN. Value: quantity of resource allocated in the given interval — a decision is basically an activity, an amount of allocated resource in a time interval. Update Function: the resource profile is modified by taking into account this allocation. Outside the specified interval the profile is not affected.

So far, we have defined components in isolation. When components are put together to model a real domain they cannot be considered as reciprocally decoupled, rather we need to take into account the fact that they influence each other’s behavior.

In OMPS, it is possible to specify such intercomponent relations in what we call a domain theory. Specifically, given a set of components, a domain theory is a function $f^{DT}$ which defines how decisions taken on one component affect the behaviors of other components. Just as a consistency feature $f^C$ describes which behaviors are allowed with respect to the features of a single component, the domain theory specifies which of the behaviors belonging to all modeled components are concurrently admissible.

In practice, a domain theory is a collection of synchronizations. A synchronization essentially represents a rule stating that a certain decision on a given component (called the reference component) can lead to the application of a new decision on another component (called target component). More specifically, a synchronization has the form $(C_i, V) \rightarrow (C_j, V’, R)$, where: $C_i$ is the reference component; $V$ is the value of a component decision on $C_i$ which makes the synchronization applicable; $C_j$ is the target component on which a new decision with value $V’$ will be imposed; and $R$ is a set of relations which bind the reference and target decisions.

The fundamental tool for defining dependencies among component decisions are relations, of which OMPS provides three types: temporal, value and parameter relations.

Given two component decisions, a temporal relation is a constraint among the temporal elements of the two decisions. A temporal relation among two decisions $A$ and $B$ can prescribe temporal requirements such as those modeled by Allen’s interval algebra [1], e.g., A EQUALS B, or A OVERLAPS $[lb,ub]$ B.

A value relation between two component decisions is a constraint among the values of the two decisions. A value relation among two decisions $A$ and $B$ can prescribe requirements such as A EQUALS B, or A DIFFERENT B (meaning that the value of decision $A$ must be equal to or different from the value of decision $B$).

Lastly, a parameter relation among component decisions is a constraint among the values of the parameters of the two decisions. Such relations can prescribe linear inequalities between parameter variables. For instance, a parameter constraint between two decisions with values “available(‘antenna’, ‘bandwidth’)” and “transmit(‘bitrate’)” can be used to express the requirement that transmission should not use more than half the available bandwidth, i.e., $\text{'bitrate'} \leq 0.5 \cdot \text{‘bandwidth’}$.

Component decisions and relations are maintained in a decision network: given a set of components $C$, a decision network is a graph $(V,E)$, where each node $\delta{C} \in V$ is a component decisions defined on a component $C \in C$, and each edge $(\delta{C_i}, \delta{C_j})$ is a temporal, value or parameter relation among component decisions $\delta{C_i}$ and $\delta{C_j}$.

The characterizing feature of decisions which define an initial condition is that these decisions do not lead to application of the domain theory. Conversely, decisions representing goals directly or indirectly entail the need to apply synchronizations in order to reach domain theory compliance. This mechanism is the core of the OMPS planning process, the description of which is outside the scope of this paper.

### 2.3 Timelines in OMPS

OMPS’s solving strategy is based on the notion of timeline. A timeline is defined for a component as an ordered sequence of its values. A component’s timeline is defined by the set of decisions imposed on that component. Timelines represent the consequences of the component decisions over the time axis, i.e., a timeline for a component is the collection of all its behaviors as obtained by applying the $f^U$ function given the component decisions taken on it. Figure 1 illustrates how a timeline for a state variable component results from a simple network of decisions and relations on that component.

The overall solving process implemented in OMPS is composed of three main steps, namely domain theory application, timeline management and solution extraction. Space requirements do not allow us to describe these processes in this paper, for the details of which we refer the reader to [5]. Suffice it to mention that timelines are the fundamental object on which OMPS’s solving process operates. Indeed, timeline management consists essentially in two types of deduction, namely
Figure 1: Three component decisions on a state variable, and the resulting earliest start time (EST) timeline.

(1) adding component decisions to timelines in order to repair flaws, and (2) performing resource and state variable scheduling to eliminate conflicts between component decisions.

A flaw is a segment of time in which no decision has been taken, thus the state variable within this segment of time is not constrained to take on certain values, rather it can, in principle, assume any one of its allowed values. The timeline in the figure contains a flaw in the interval $[30, 40]$. Dedicated timeline management procedures implement the process of deciding which value(s) are admissible with respect to the state variable’s internal consistency features (i.e., the component’s $f^C$ function).

The nature of inconsistencies, on the other hand, depends on the specific component we are dealing with. In the case of state variables, an inconsistency occurs when two or more value choices whose intersection is empty overlap in time. In the example above, this occurs in the interval $[0, 10]$. As opposed to flaws, inconsistencies do not require the generation of additional component decisions, rather they can be resolved by posting further temporal constraints. Also this process is carried out during timeline management, and employs dedicated scheduling procedures.

3. The Case#1 Problem

Currently, long, medium and short term planning for Mars Express is carried out through a collaborative problem solving process between the PST (Payload Support Team, ESTEC) and the MPG (Mission Planning Group, ESOC). These two groups of human planners iteratively refine a plan containing all mission activities for Mars Express. The process starts at the long term level, and is gradually refined to obtain fully instantiated activities at the short term level. This process continuously leads to weekly STPs, which are then further refined every two days to produce final executable plans.

The goal of Case#1 is to develop a pre-planning optimization tool for science operations planning within the Mars Express mission. Specifically, Case#1 focuses on the generation of a pre-optimized skeleton LTP which will then be subject to cooperative PST/MPG refinement.

We observe that at the first step of the negotiation process (between PST and MPG) the main source of approximation comes from the fact that the MPG has no information with reference to the Mars Express Payload Operation Requests (i.e., science operations, PORs). The reference to the PORs is given only when the PTR (Pointing Timeline Request file) is issued by the PST on the basis of the input skeleton plan. Indeed, these science requests often require the satellite to point to the planet, reducing its ability to obtain energy from the Sun and to send data back to Earth. Also, science operations consume power and exclude the possibility of performing maintenance operations. On the other hand, the MPG requires the spacecraft to perform maintenance and other service manoeuvres in order to maintain the spacecraft operational. Overall, the PST and the MPG have to cooperatively converge on a plan which resolves the complex interplay between science, pointing direction, power, data transmission and maintenance operations.

In this context, the challenge of APSI is to provide an automated procedure for producing a “good” skeleton plan, i.e., a LTP that takes into account the needs of both parties, thus reducing the effort in reaching a medium-term plan. Overall, the generated LTP should be such that: (a) the number of (expensive) iterations between PST and MPG is reduced; (b) a set of objective functions (e.g., total volume of data for down-link operations; the number of pericentres for science operations; the number of uplink windows) is maximized.

3.1 ESA Requirements and Current Practice

For each orbit, baseline operations are split into three phases. Around pericentre, around apocentre, and between pericentre and apocentre phases. Around pericentre, the spacecraft is generally Nadir pointing. This allows close observation of the Martian surface. Between pericentre and apocentre passages, the spacecraft is either Earth pointing for transmission of scientific data down to Earth, or is performing a maintenance operation. Communication with Earth must occur within a ground station availability window. Ground station visibility can also partially overlap or fully contain a pericentre passage. Maintenance operations, on the other hand, should occur around an apocentre passage.

Given these requirements, an initial skeleton plan for Case#1 is currently generated by the MPG by allocating over the planning horizon (which covers hundreds of orbits) three different kinds of decisions:

- the selection of the maintenance windows (generally centered around the apocentres and used primarily for momentum wheel-offloading – WOL);
- the selection of the communication windows (ground stations de-overlapping);
- the selection of the windows for science operations (generally positioned around the pericentres, although it is possible to perform so-called high altitude observations, which can be executed up to 2000
In the current model for Case#1, these windows have a fixed minimal and maximal duration. We observe that the decision process for windows selection has several branching points: as to which pericentres are used for uplink/downlink communications or science operations; which ground station is used among the available ones; which apocentres are used for spacecraft orbit maintenance activities (primarily momentum wheel-offloading) or communication. Roughly speaking such windows cannot overlap one another.

There are many further hard and soft constraints to satisfy. Constraints on uplink windows frequency and duration require four hours uplink time for each 24 hours (hard constraint). However, it is preferable to split a four-hour uplink window in two two-hour uplink windows. Apocentre slots for spacecraft maintenance windows must be allocated between 2 and 5 orbits apart, and the duration is 90 minutes centered around the apocentre time. In addition, the sequences 2-2-5 and 5-5-2 of orbit gaps are not allowed. There are constraints on the maximal value of the power requested to the satellite power subsystem and there is a constraint on the maximum allowed Depth-of-Discharge (DoD) value of the on-board lithium batteries. Also communication activities are source of many hard temporal constraints. For example: minimum/maximal durations for the X-band transmitter in the on state; minimum duration for the X-band transmitter in the state off; periods when the X-band transmitter has to be off (e.g., eclipses, occultations, slewing manoeuvres and non-Earth pointing status); offset times with respect to some trigger events (e.g., the offset timed for the start/end times of a downlink window respect to the acquisition of signal/loss of signal trigger events); station hand-over times between two adjoining stations (e.g., 5 minutes physical gap between contacts).

In addition, there are preferences (soft constraints) for ground station selection (de-overlapping). Ground stations have different features: have different dish diameters (there are 70 meters dish antennas, 35 meters and 34 meters ones); generally they allow both uplink and downlink communications, but there are cases where it is only possible to uplink; there are ESA ground stations and DSN (Deep Space Network) ones from NASA. Roughly speaking, the rule is to try to use 70 meters dish antennas from ESA when possible, and move to other choices when there are negative consequences on the maintenance or science activities (e.g., an high priority experiment cannot be carried out on a given pericentre).

3.2 Case#1 as an Optimization Problem

We define the problem related to Case#1 as a timeline based optimization problem as follows.

**Decomposition into components**. We consider two different types of components: (1) **Controllable Components**, which define the search space of the problem, and whose timelines ultimately represent the solution to the Case#1 problem; (2) **Uncontrollable Components**, representing values imposed over time which can be only observed.

**Modeling ESA requirements**. ESA requirements are modeled in a domain theory composed of components and synchronizations among components. The domain theory is expressed in a general modeling language, called DDL.3, which allows to describe components, their consistency features and the synchronizations among decisions on these components.

**A feasible solution**. A solution to the Case#1 problem is a set of timelines for the controllable components, each of which is (a) consistent with respect to the component’s consistency features, (b) complete (i.e., without flaws), and (c) consistent (i.e., there exists no interval of time in which conflicting decisions overlap).

The specific components employed for Case#1 are described in the following.

**Controllable Components**. These components model the spacecraft’s pointing mode, science and maintenance operations, the spacecraft’s ability to communicate, as well as the spacecraft’s power management module. Specifically, we model the five controllable components:

**Science**. A state variable type component whose allowed values represent the possible science operations that can be performed. The DDL.3 specification of this component’s consistency features is shown in figure 2. Notice that it specifies (1) the path of the Java class which implements the component (first line), (2) the allowed transitions between values, and (3) the minimum and maximum durations of each value (in this case, \([1, +\infty]\)). The corresponding timed automaton is illustrated in figure 3. We omit the DDL.3 specification of the remaining three state variable type components in this domain.

```java
COMP_TYPE PST. {...}, StateVariable SC_SCIENCE
  (Nadir_Science(), Radio_Science(), Inertial_Science(), No_Science())

VALUE No_Science() {1, +INF}
  MEETS { No_Science(), Radio_Science(), Inertial_Science(), Nadir_Science() }
VALUE Nadir_Science() {1, +INF}
  MEETS { No_Science() }
VALUE Radio_Science() {1, +INF}
  MEETS { No_Science() }
VALUE Inertial_Science() {1, +INF}
  MEETS { No_Science() }
```

Figure 2: The DDL.3 specification of the Science component and its consistency features.

**Pointing**. A state variable type component whose allowed values represent the possible pointing modes of the spacecraft. The allowed values and possible
transitions (consistency features) resulting from the DDL.3 specification (not shown) of this component are illustrated in figure 4.

**Maintenance.** A state variable type component whose allowed values and transitions model the possible maintenance modes of the spacecraft. These include: WOL() (wheel-offloading), No_Maintenance(), Orbit_Manoeuvre() and Idle().

**Communication.** A state variable component modeling the communications subsystem of the spacecraft, whose possible values and consistency features are shown in figure 5.

**Power management.** The spacecraft’s power management subsystem is integrated into the overall infrastructure as an ad-hoc component of type PowerManagement. This component is implemented as an extension (in the Java sense of the word) of a reusable resource. Thus, in DDL.3, the specification of this component is reduced to:

```plaintext
COMP_TYPE PST.:[...].Resource.PowerManagement
SC_POWER_MANAGEMENT :
[900,[486000000,4860000000],1];
```

which states the salient features necessary for instantiating a component of type PowerManagement, namely the maximum power that can be provided (900 Watts), the minimum and maximum levels of charge (max.min) of the on-board batteries (486,000,000 and 4,860,000,000 Watts/millisecond in our case), and a constant $\alpha$ which approximates the charging profile (1 in our case). All the other details of the component (e.g., how power request and production reflect on the battery’s level of charge, etc.) are directly programmed into the component. More specifically, the component’s behavior is a temporal function $bat(t)$ representing the battery’s level of charge. Assuming that power consumption decisions resulting from other components are described by the function $cons(t)$, the update function calculates the consequences of power production ($sf(t)$) and consumption on $bat(t)$ as follows:

$$
bat(t) = \begin{cases} 
  L_0 + \alpha \int_0^t (sf(t) - cons(t)) dt 
  & \text{if } L_0 + \alpha \int_0^t (sf(t) - cons(t)) dt \leq \text{max}; \\
  \text{max} & \text{otherwise}.
\end{cases}
$$

where $L_0$ is the initial charge of the battery at the beginning of the planning horizon.

**Uncontrollable Components.** In addition to the five components above, we instantiate four other state variable type components for maintaining contingent events such as orbit events and communication opportunity windows. More specifically, we model:

**Orbit events.** A state variable type component which maintains the pericentres and apocentres of the spacecraft’s orbit. These are component decisions of value Pericentre() or Apocentre(), and are fixed in time according to the information found in ESA’s orbit events file.

**Station Allocation.** Three state variables for maintaining the visibility of three ground stations (MAD, GDS and NNO). These components have allowed values \{Available(?rate,?ul,?antennas), Unavailable()\}, where the ?rate parameter (of numeric type) indicates the bitrate at which communication can occur, ?ul (of enumeration type) indicates whether the station is available for upload, download or both, and the ?antennas parameter (of enumeration type) indicates which dish is available for transmission.

In summary, we have a total of nine components, five of which (the controllable components) are subject to the imposition of decisions during the planning process. All components are instantiated from the component types described above, e.g.:  

```plaintext
COMPONENT SCIENCE:SC_SCIENCE;
COMPONENT POWER_MANAGEMENT:SC_POWER_MANAGEMENT; [...] 
```
Synchronizing components with a Domain Theory. So far, we have summarized the individual components and their consistency features. We now turn our description how components are related in the domain theory. As mentioned, we employ synchronizations to model the required dependencies among components. These synchronizations are described in the domain theory in DDL.3 syntax. For instance, the following snippet contains a synchronization describing the requirements of science operations on pointing, communications and maintenance:

```plaintext
COMPONENT SCIENCE:SC_SCIENCE {
  VALUE Nadir_Science() {
    EQUALS POINTING_MODE Nadir(),
    EQUALS COMM TTX_OFF(),
    EQUALS MAINTENANCE No_Maintenance()
  }
  VALUE Inertial_Science() {
    EQUALS POINTING_MODE Inertial(),
    EQUALS COMM TTX_OFF(),
    EQUALS MAINTENANCE No_Maintenance()
  }
  VALUE Radio_Science() {
    EQUALS POINTING_MODE Earth_Comm(),
    EQUALS COMM TTX_OFF(),
    EQUALS MAINTENANCE No_Maintenance()
  }
};
```

The synchronization specifies that all science modes require the pointing state variable to assume a certain pointing mode value (respectively, Nadir(), Inertial() and Earth_NonComm()), all communication system transmissions should be off, and that maintenance cannot be performed while doing science. All three synchronizations specify that the required values on the target state variables are subject to an EQUALS temporal constraint. This implies, for instance, that if the planner decides to expand the domain theory by following the above synchronization on a Nadir_Science() decision, it will impose a Nadir() decision on the Pointing component and bind it with an EQUALS constraint to the Nadir_Science().

Other synchronizations in the domain theory describe power requirements. For instance, the fact that the spacecraft’s pointing manoeuvres require certain amounts of power is modeled as a synchronization on the Power Management component as follows:

```plaintext
COMPONENT POINTING_MODE:SC_POINTING_MODE {
  VALUE Slew() {
    %Thermal -- see MEX-ESC-TN-5606
    EQUALS POWER_MANAGEMENT 95
  }
  VALUE Earth_No_Comm() {
    %Thermal -- see MEX-ESC-TN-5606
    EQUALS POWER_MANAGEMENT 95
  }
  VALUE TTX_DL_UL_ON() {
    %Assuming X-band transm. -- see MEX-ESC-TN-5606
    EQUALS POWER_MANAGEMENT 120,
    EQUALS POINTING_MODE Earth_Comm(),
    EQUALS SCIENCE No_Science(),
    EQUALS MAINTENANCE No_Maintenance(),
    EQUALS NO_GS Available(?rate,?availability, ?antenna),
    ?availability = dl
  }
};
```

Another example of complex synchronization is the following, where the Communications component is related to power consumption and other requirements on pointing, maintenance and station availability:

```plaintext
COMPONENT COMM:SC_COMM {
  VALUE TTX_DL.ON() {
    %Assuming X-band transm. -- see MEX-ESC-TN-5606
    EQUALS POWER_MANAGEMENT 120,
    EQUALS POINTING_MODE Earth_Comm(),
    EQUALS SCIENCE No_Science(),
    EQUALS MAINTENANCE No_Maintenance(),
    EQUALS NO_GS Available(?rate,?availability, ?antenna),
    ?availability = dl
  }
  VALUE TTX_UL.ON() {
    %Thermal -- see MEX-ESC-TN-5606
    EQUALS POWER_MANAGEMENT 95
  }
};
```

where the final line ?availability = dl indicates that the ?dl parameter has to be assigned value dl, reflecting that transmission can occur only for data download to Earth.

Case#1 is in principle a multi-objective optimization problem involving a trade-off between multiple conflicting and non commensurate objectives (e.g., total downlink data volume vs. the number of high priority science pericentres). At this stage of the work, we are considering both a multi-objective formulation and a single-objective one.

### 3.3 Case#1 Software Prototype

The APSI project is currently nearing the end of its first year. The expected outcome is a full implementation, based on the OMPS software framework, of a mission planning support system for Mars Express (Case#1). The problem formulation described above is the first Case#1 prototype produced in the project, and was used in a first rapid user evaluation cycle aimed at refining the architecture and completing knowledge elicitation from the end users of the system.

The implementation of the prototype demonstrates two key points. First, the advantage of employing an AI model-based approach, in which the relevant features of the domain are described in a high-level formalism (DDL.3), allowing to harness the versatility of general modeling and solving capabilities. The second feature which enables fast prototyping is the capability of designing ad-hoc components and including them in the domain specification.

In more quantitative terms, the overall effort behind the implementation of the OMPS framework consists in 240 classes (about 35K LOC\(^2\)). An additional 26 classes (i.e., consisting in just over 10% of the total project’s source code) were implemented for Case#1. Specifically, the realization of the Case#1 prototype consists in the following:

\(^2\)Lines of Code.
Additional function implementation. An additional component for integrating power management functionality, the PowerManagement class. This component essentially extends the general purpose ReusableResource component, in that it adds an ad-hoc update function to calculate the battery consumption profile as shown above. As a consequence, the heavily tested and optimized scheduling and constraint propagation functions implemented in the ReusableResource component are re-used within the context of power management, thus contributing to the ease of prototype development. Overall, the implementation of the PowerManagement component consists in a mere 251 LOC.

Algorithm tuning. The current implementation of the Case#1 prototype employs a specialization of the generic OMPs main solving strategy. This consists in a pre-processing step which places some initial science and maintenance operations around, respectively, pericentres and apocentres.

Input. A series of parsers for reading relevant ESA mission files and instantiating the component decisions and relations which make up the timelines of the uncontrollable components (orbit events, ground station availability and solar flux).

Model. A DDL.3 domain model in which the necessary components are declared, their consistency features defined and the synchronizations that govern the domain theory are formulated. In the first prototype, this model contains nine component definitions and 14 synchronizations.

4. Conclusions

Our experience with ESA began with the development of MEXAR2 [4] and RAXEM [3], two operational mission software tools which are currently in active use within Mars Express. These previous results were conceived as ad-hoc software tools, and were essentially developed following a software development life-cycle loosely based on the waterfall approach (understanding the problem, designing a solution, building, testing and implementing the solution). In APSI, the use of the OMPs framework provides two advantages over “classical” software development approaches: (1) rapid prototyping of new targeted software applications (e.g., Case#1), and (2) it enables short fixed-time interaction with the users in order to refine the requirements within an extreme programming development style. The key factors that enable this form of software development are, on one hand, the AI model-based approach, and on the other the possibility of designing and implementing the fundamental properties of the application domain into separate, extendable and re-usable components.

The Case#1 experience has thus far been very positive. In a few weeks, we were able to prototype a software system for our users (MPG) through which we received useful and constructive feedback. The above considerations on the realization of the first prototype suggest a general schema for implementing domain-specific decision support tools, namely: (1) implement any additional component types that are required by the application context; (2) extend and/or tune the general solving procedure where necessary; (3) define a DDL.3 model where the component types are instantiated with domain-specific characteristics and are logically bound by synchronizations. Of course, these three steps cannot alone provide a complete deployable software tool. Nonetheless, they provide a means to reduce the gap between prototype and final application by factoring away all the major algorithmic and modeling design choices.

5. References