Unifying Planning and Scheduling as Timelines in a Component-Based Perspective

Simone Fratini*, Federico Pecora† and Amedeo Cesta

ISTC-CNR, National Research Council of Italy
Via San Martino della Battaglia 44, I-00185 Rome, Italy
{name.surname}@istc.cnr.it

Abstract

The timeline-based approach to planning represents an effective alternative to classical planning for complex domains requiring the use of both temporal reasoning and scheduling features. This paper discusses the constraint-based approach to timeline planning and scheduling introduced in OMPS. OMPS is a an architecture for problem solving which draws inspiration from both control theory and constraint-based reasoning, and which is based on the notion of *components*.

The rationale behind the component-based approach shares with classical control theory a basic modeling perspective: the planning and scheduling problem is represented by identifying a set of relevant domain components which need to be controlled to obtain a desired temporal behavior for the entire system. Components are entities whose properties may vary in time and which model one or more physical (or logical) domain subsystems relevant to a given planning context. The planner/scheduler plays the role of the controller for these entities, and reasons in terms of *constraints* that bound their internal evolutions and the desired properties of the generated behaviors (goals).

Our work complements this modeling assumption with a constraint-based computational framework. Admissible temporal behaviors of components are specified as a set of causal constraints within a rich temporal specification, and goals are specified as temporal constraint preferences. The OMPS software architecture presented in this paper combines both specific and generic constraint solvers in defining consistent timelines which satisfy a given set of goals.

*Corresponding author: Simone Fratini, ISTC-CNR, Via San Martino della Battaglia 44, I-00185 Rome, Italy, phone: +39 06 44595 270, fax: +39 06 44595 243, e-mail: simone.fratini@istc.cnr.it.

†Federico Pecora is currently at the AASS Mobile Robotics Lab, Dpt. of Technology, Örebro University (Sweden).

Contents

1 Introduction

Problem solving systems can be classified into two main categories: numeric-mathematical and symbolic. Classical control systems are an example of numeric-mathematical problem solving systems, while AI planning and scheduling systems are examples of symbolic decision makers. Symbolic decision makers are usually more flexible than numeric-mathematical systems. On the flip side of the coin, symbolic planning is marred with computational issues (see [37] for a wider discussion about connections between AI planning and control theory).

The timeline-based approach models the planning and scheduling problem by identifying a set of relevant *features* of the planning domain which need to be controlled to obtain a desired temporal behavior. Timelines model entities whose properties may vary in time and which represent one or more physical (or logical) subsystems which are relevant to a given planning context. The planner/scheduler plays the role of the controller for these entities, and reasons in terms of *constraints* that bound their internal evolutions and the desired properties of the generated behaviors (goals).

Current AI planning literature shows that timeline-based planning can be an effective alternative to classical planning for complex domains which require the use of both temporal reasoning and scheduling features (see [35, 34, 10, 28, 21, 39]). The constraint-based approach is shown to be an efficient framework for modeling and solving both temporal and resource allocation problems (see [20, 38, 41, 7, 11]). We therefore propose OMPS (short for Open Multi-component Planner and Scheduler), a timeline-based planning and scheduling system which draws inspiration from both control theory and constraint-based reasoning. In this paper, we present both the theoretical framework behind OMPS, the *component-based* approach where we extend the idea of timeline by introducing the concept of *component*, and the architecture of OMPS.

This paper is organized as follows: Section 2 connects our work within the current literature. Section 3 introduces more formally our approach presenting a definition of *component-based* planning and scheduling, and explaining the key features through a toy example. Sections 4 presents a complete report on OMPS timeline-based planning algorithms. Section 5 presents OMPS as a software architecture, focuses on the constraint-oriented design of the various modules, and shows a more concrete example from a realistic domain in space mission planning. Conclusions end the paper.

2 Planning, Scheduling and Constraint Reasoning

The aim of this section is to give a brief survey of the relationship between planning, scheduling and constraint reasoning.

A Constraint Satisfaction Problem, or a CSP, consists of a set of variables $X = \{X_1, X_2, \ldots, X_n\}$ each associated with a domain D_i of *values*, and a set of constraints $C = \{C_1, C_2, \ldots, C_m\}$ which denote the legal combinations of values for the variables s.t. $C_i \subseteq D_1 \times D_2 \times \cdots \times D_n$. A solution consists of assigning to each variable one of its possible values so that all the constraints are satisfied. The resolution process can be seen as an iterative search procedure where the current (partial) solution is extended

at each cycle by assigning a value to a new variable. As new decisions are made during this search, a set of *propagation rules* removes elements from the domains D_i which cannot be contained in any feasible extension of the current partial solution (see [42, 16] for further details).

While CSPs are now widely used and accepted by the scheduling community (see [4] for instance), constraint-based reasoning had less impact in planning. The principal cause of its success in scheduling (and, dually, of its less extensive impact in planning) resides in how the two problems are defined.

The scheduling problem concerns the allocation of a known (at least in the classical approach, e.g., [8]) set of activities to available resources. The formulation of the problem requires modeling capacity and temporal (or simple precedence) constraints among the activities, making the CSP suitable for representing and solving scheduling problems.

On the contrary the planning problem, from a very general point of view, is concerned with the generation of a set of tasks to achieve a given goal (see [27] among others for a more comprehensive discussion). The tasks to be planned are not know in advance, and this has made it more difficult to exploit classical constraint-based formulations, where variables and constraints must be specified before solving the problem (see also [6]). This has steered research toward more sophisticated models like DCSPs (Dynamic Constraint Satisfaction Problems) [33].

The following sections discuss planning, scheduling and some schemas of their integration highlighting (when appropriate) the role of constraint-based reasoning.

2.1 Action- vs. Timeline-Based Perspectives in Planning

Many planning systems have been studied and proposed: basically the only things they have in common is the general task of coming up with a partially ordered sequence of decisions, legal with respect to a set of rules (called a *domain theory*), that will achieve one or more goals starting from an initial situation. However, the analogies do not go further: there have been a lot of differences in the formalism chosen to represent the world in which the planner performs its task, in the general conception about what a plan is and about how the planning process is performed. Different problems have been addressed through different approaches: in *temporal planning* for instance, the focus is on deciding plans by explicitly managing complex temporal information, such as durations and separations between actions. More in general, it is possible to distinguish two distinct approaches to the planning problem: in the first one, called here *action-oriented*, the world is seen as an entity that can be in different *states*, and in which a state can be changed by performing *actions* (*a-la* STRIPS[19]). The domain theory specifies which rules must be followed when actions are performed, i.e., where they can be applied and how the world state is modified as a consequence of the actions. The planning problem is to find a partially ordered sequence of actions that, applied to an initial world state, leads to a final state (the goal) eventually satisfying several conditions about which sequence of states the world must go through (often called *extended goals*). This is the most classical and commonly accepted idea of what planning is.

In the second approach, named here *timeline-based*, the world is modeled in terms of a set of functions of time that describe its evolution over a temporal interval. These

functions change by posting planning *decisions*. The domain theory specifies what can be done to change these evolutions, and the task of the planner is to find a sequence of decisions that bring the entities into a final situation that verifies several conditions. By analogy, we can call these conditions goals. This approach, which is more recent than the more "classical" action-based approach, is evolving rather rapidly due to the fact that it is well suited for modeling and solving non-trivial real world problems.

In *action-oriented* planning the solver searches a space of *world states* to find one in which a given goal is achieved. For any world state we assume that there exists a set of applicable operators that can transform the state into another state. The task of the problem solver is to find some composition of operators that transform a given initial world state into one that satisfies a set of stated goals. A distinctive feature of this form of planning is a clear distinction between *actions* and *states*. Actions are performed by a hypothetical executing agent to change the world's state in order to achieve its goals. By contrast, in timeline-based planning the world is modeled as a set of entities, often called *state variables*, that describe the state of the world. Unlike the action-oriented approach, where the state of the world is changed by means of actions, here the state is changed more "directly" by posting planning decisions that force the transition of the state of the world. In this form of planning, there is no explicit notion of action, rather the focus is on the state of the world itself. Dually, the plan is not a sequence of actions, but a sequence of transitions among states of the world. In timeline-based planning, the result of the planning process is called a *timeline*, as it represents an evolution in time of the states of the physical system(s) modeled in the domain.

In action-based planning, the planner decides actions whose preconditions are true in state S_i to obtain a state of the world S_j where the postconditions of the action are true. Roughly speaking, in the timeline-based approach (see figure 1), the planner decides which states the world should be in during given segments of the timeline. It is on this temporally-grounded representation of "what is happening" in time that the planner operates during planning. The timeline representation, in short, allows the planner to apply further decisions based on the consequences that these decisions have on the complete planning horizon. The progressive propagation of decisions determines how the entire timeline is affected (thus, as stated, the state is changed more "directly"). As in action-based planning, the planner employs a domain theory to ascertain which decisions to impose. However, while in action-based planning the domain theory describes operators which change the state of the world, in timeline-based planning the domain theory represents how different decisions should be synchronized. This is represented by the notion of synchronization (see the $s(i, j)$ edges in the figure). Synchronizations describe the constraints that are to be imposed on the overall timeline when a specific decision is present. For instance, a synchronization may state that a decision to "navigate to a destination" needs to be synchronized with a decision to "consume a certain quantity of fuel". As opposed to action-based planning, the focus here is on the state rather than on the operator needed to change the state.

An important point to make in this distinction is the fact that both approaches can, to different extents, take into account time. In action-based planning this has been done by enhancing planners with the ability to reason upon actions with durations (notice that "classical" planning represents the case in which all actions have unit durations). In order to to this, action-based planners typically need to incorporate specialized heuris-

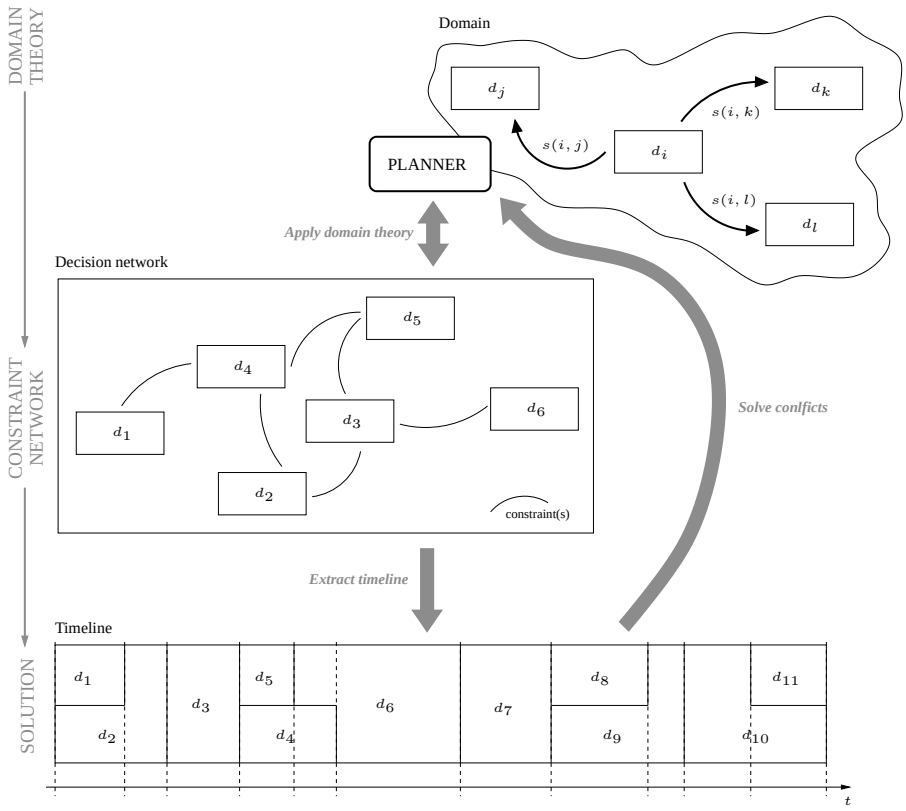


Figure 1: Principle behind timeline-based planning in OMPS.

tics to take into account the overall effect of an action's duration. In contrast, timeline based planners maintain a representation of the consequences of the decisions in time (the timeline), along with the associated temporal constraints among the decisions. This is a common feature between timeline-based planning architectures in use today (among which RAX-PS [28] and its successors EUROPA and EUROPA2, that inherit the experience of HSTS [35, 34] and ASPEN [14]). The fact that all decisions are propagated in time is what allows OMPS to ascertain, in the face of every decision taken, whether the temporal constraints (and, as we will see, any other constraints for which a dedicated propagation or solving procedure is implemented) are satisfied.

One advantage of timeline-based planning thus lies in its "natural" predisposition for temporal reasoning. OMPS builds a solution to the planning problem by constructing and maintaining a network of decisions (the middle layer in figure 1). The nodes of the network are decisions and the edges are constraints that bind these decisions. These constraints are imposed as a consequence of the planner taking into account a particular synchronization in the domain theory. When the planner decides to impose a decision, it adds it to the network along with all the necessary other decisions specified in the domain theory and imposes the associated constraints. Typically, timeline-based

planners provide the possibility to specify temporal constraints in the synchronizations. Since temporal constraint propagation is polynomial ([15]), these planners can afford to propagate the constraints thus identifying temporal inconsistencies early on, and backtracking on decisions. Yet the temporal admissibility of the constraint network may not guarantee the existence of an executable plan if other types of relations are taken into account in the planning problem. This is the case in OMPS, which is capable of dealing with capacity-bounded resource constraints as well as linear inequalities among parameters of decisions. The admissibility of the current timeline with respect to resources and parameters is taken into account by other specialized propagation and resolution algorithms. For resources, for instance, OMPS employs profile-based scheduling to resolve resource conflicts. This results in the addition of further constraints to the network in order to "separate" the decisions whose temporal overlapping over-consumes a resource[1]. In order to verify whether further reasoning is necessary to obtain an admissible solution, the planner extracts the current timeline from the decision network and inspects it for inconsistencies. Indeed, this is why the approach is named "timeline-based": the planner observes the timeline to ascertain whether a feasible plan can be achieved with the current decisions, and this verification leads the planner to trigger specialized reasoning algorithms, which in turn can result in new constraints and/or decisions.

It is easy to appreciate already from this brief and incomplete summary of action- and timeline-based planning that the former approach is particularly suited for modeling and reasoning about logical consequence: an action-based domain theory provides the proper level of abstraction as well as the most effective metaphor for describing highly complex causal theories, e.g., a freecell game or the well known blocks world. Conversely, expressing such concepts through "decision templates", although formally equivalent entails non-trivial modeling, as preconditions and effects need to be modeled as synchronizations among decisions. On the other hand, modeling time comes very natural to the timeline-based paradigm: decisions can have interval durations and synchronizations can prescribe non-trivial temporal requirements (e.g., Allen's interval algebra [1]). In addition, in OMPS both temporal and resource reasoning is tightly incorporated into the planning algorithm. This is due to its strongly constraint-oriented nature: OMPS reasons within the "sandbox" of the decision network, which can easily represent both the planning and the scheduling problem by means of a constraint based representation which is natively supported by the best and most mature temporal and resource reasoning technology available.

It is useful to compare the strongly constraint-oriented approach of OMPS planners with approaches involving constraint reasoning in action-based planning (see, e.g., [29, 30, 25, 43, 17]), where the planning problem (or a sub-part of a planning problem) is formulated a set of constraints over some variables, a solution of the CSP is found using a state-of-the-art CSP solver and the solution is re-translated into a partially ordered set of actions. Unlike these planners, the timeline-based approach has a more strict connection with CSP techniques, as it explicitly represents temporal information or resource reasoning, features that have a successful CSP-based tradition. Timeline-

[1] Specifically, this is called the *Precedence Constraint Posting* approach ([40, 13, 36, 11]). It is a widely used for resource-constrained scheduling, and consists in posting a sufficient set of additional precedence constraints between pairs of activities that contend the same resource, to ensure resource feasibility.

based planning leverages the CSP model to a larger extent, and in some systems the CSP approach is so pervasive that the modeling language used is CSP-based as well (meaning that the planning domain is described by means of constraints instead of actions with pre-conditions and effects). In OMPS for instance, the domain theory is formulated using constraints and the solving process is a synergy of CSP solving processes over different groups of variables and constraints.

It is worth mentioning here other systems that exploit CSP technology to model and solve one or more features of the planning problem, such as PARCPLAN [32, 18] (where resource reasoning is implemented as a CSP), IXTET [26] (this is the architecture that more closely follows a CSP approach as a general frame of reference for temporal and resource reasoning), HSTS [35, 34] (designed with a constraint-based scheduling approach). Finally, some works leverage the idea of using constraint-based reasoning to an even greater extent, e.g., [21, 22] and OMP [9] (where the CSP approach permeates almost the entire design and solving process).

2.2 OMPS objectives

In comparison with all the above mentioned constraint- and timeline-based planners, OMPS addresses the following relevant novelties:

- It introduces the concept of *components* as a means to encapsulate the reasoning power of dedicated solvers and/or the complexity of non-symbolic state maintenance.
- It uses the concept of *decision network*, a constraint network of decisions on different components which the planner leverages to propagate new decisions and constraints that are synthesized during planning.

The introduction of components as first-class citizens is aimed at generalizing the work of both time-line based planners and constraint-based schedulers. Both these types of systems produce results in terms of temporal functions, namely legal sequences of temporal tokens in state-variables *a-la* EUROPA and consistent temporal resource profiles in state of the art schedulers. In capturing this similarity we have designed a generalization aiming at modeling potentially different types of temporally evolving features under the same abstraction, called *components*. As we show in this paper, components are a generalization of concepts such as state-variables and different types of resources. Indeed, components can accommodate these as well as other entities, so long as they maintain the same external interface.

Furthermore by leveraging the fact that all decisions are maintained in a constraint network, we seamlessly introduce the capability to take into account constraints which do not necessarily involve time or resource usage. This is achieved through the introduction of dedicated solvers for propagating and solving subproblems involving these constraints, such as general purpose CSP solvers.

It is worth observing that the whole component-based approach shares common elements with control theory. As we describe in detail in this paper, components (such as a state variable or a resource) are objects with their own internal behavior, and the problem we address in this type of planning is to find an input sequence that can be

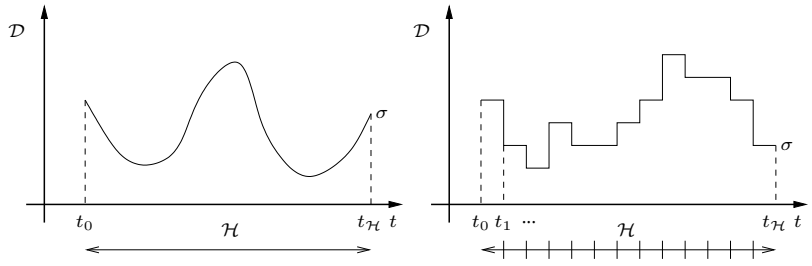
used to obtain desired outputs. Additionally, the notions of *time* and *concurrency* are embedded into this kind of model of the world. Component based modeling as it is done in OMPS allows to model the world as separate and interconnected entities that concurrently behave in time as a consequence of decisions that are posted over them (see also [37] for a wider discussion about connections between AI planning and control theory).

3 The Component-Based Approach

As mentioned, the philosophy behind OMPS is derived from classical control theory, in that the planning and scheduling problem is modeled by identifying a set of relevant *components* which need to be controlled to obtain a desired behavior. Components are primitive entities for knowledge modeling, and represent logical or physical subsystems whose properties may vary in time. An intrinsic property of components is that they evolve over time, and that decisions can be taken on components which alter their evolution.

Definition 1 *A* component C *is an entity that has a set of possible temporal evolutions over an interval of time* $\mathcal{H}$*.* $\mathcal{H}$ *is the* horizon *over which these evolutions are defined. The component's evolutions over time are named* behaviors*. Behaviors are modeled as temporal functions over the temporal interval* $\mathcal{H}$*. We denote a behavior's co-domain with* $\mathcal{D}$*.*

Figure 2(a) shows an example of a component behavior in the most general case (continuous time), while Figure 2(b) exemplifies a stepwise constant behavior (instantaneous changes). In both cases, the range of the function (denoted $\mathcal{D}$) can be either a continuous or a discrete set of values.



(a) General case (b) Stepwise constant

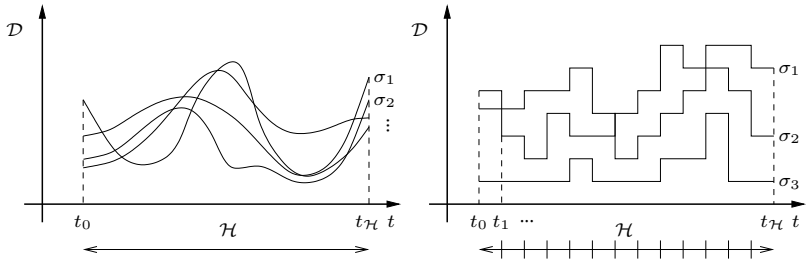
Figure 2: Examples of single behaviors.

During any interval of time in $\mathcal{H}$ a component can have one or more possible behaviors. We denote the set of possible behaviors of a component C as $\mathcal{B}(C)$. Each behavior $\sigma \in \mathcal{B}(C)$ describes a different way in which the component's properties vary in time during the temporal interval of interest (see Figure 3). However, 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. 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, i.e., those behaviors which adhere to the "physical" constraints. Such consistent behaviors are defined by a *consistency function*.

Definition 2 *A component has a* consistency function $f^C : \mathcal{B}(C) \rightarrow \{\mathrm{true}, \mathrm{false}\}$ *which determines which of all the possible component behaviors are physically possible.*

As we will see, planning for components equates to imposing planning decisions so as to achieve a desired temporal evolution of the component.In other words, in the absence of planning decisions, a component's possible evolutions are determined exclusively by the consistency function.



(a) General case (b) Stepwise constant

Figure 3: Collections of behaviors.

For instance, let us model a light bulb component. A light bulb's behaviors are functions over time in the co-domain $\{\mathrm{on}, \mathrm{off}, \mathrm{burned}\}$. 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 function f^C.

It is in general possible to represent behaviors in different ways, depending on (1) the chosen temporal model (continuous vs. discrete, or time point based vs. interval based), (2) the nature of the range (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). For instance, the consistency function of a satellite that can slew, point, take a picture and be idle can be described by a finite state machine defining the allowed transitions between these states. However, if necessary the complete control system can be used to implement the consistency function if the level of the planner's decisions needs to be more granular.

The component-based approach used in OMPS allows us to model a number of "classical" planning and scheduling concepts as components. Specifically, in order to model a classical planning problem over what are commonly known as state variables (as used for instance in [34]), we model the following component:

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 function*: 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 [3] (e.g., the three state variables in Figure 8). Note that a distinguishing feature of a state variable is that not all the transitions between its values are allowed.

Capacity-bound resources are a distinctive characteristic of many scheduling problems. In OMPS we model such scheduling problems by employing a component which models the characteristics and behaviors of such entities.

Resource. *Behaviors*: integer or real functions of time, piecewise, linear, exponential or even more complex, depending on the requirements of the domain. Each behavior represents a different profile of resource consumption. *Consistency function*: 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.

As noted above, the component-based nature of OMPS allows to accommodate complex functions for determining the consistency of a component's behavior. This opens to an interesting possibility that is novel within the planning scenario, namely that of modeling potentially complex physical systems as "gray boxes" within the planning domain. Specifically, the consistency function can implement a low-level control system which regulates the behavior of the physical component we are modeling, while predicating higher-level commands within a larger planning context. As we shall explain later, the OMPS software architecture is flexible to the point that all aspects of decision network maintenance are definable by the user. We provide an example in Section 5.1, where a battery component is achieved as an extension of a renewable resource component.

3.1 Component Decisions

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 the consistency function).

Definition 3 *A* component decision δ_C *is a pair* (d, τ) *which defines an alteration of the behaviors of a component* C. d *is the* value *of the decision, and* τ *is the* temporal element *over which the component is to assume the given value.* τ *can be:*

- *A time instant (TI)* t *representing a moment in time.*
- *A time interval (TIN), a pair of TIs defining an interval* $[t_s, t_e)$ *such that* $t_e \geq t_s$.

It is worth noticing that different decisions can be taken over the same type of component. The way the component's behaviors are affected by a decision is modeled in an *update function*:

Definition 4 *An* update function *for a component C is a function which determines how the component's behaviors change as a consequence of the decisions that are imposed on it. If $\mathcal{B}(C)$ is the set of behaviors for the component, given a decision δ_C on the component C, the update function "updates" the set of possible concurrent behaviors $2^{\mathcal{B}(C)}$ appropriately, i.e., $f^U : 2^{\mathcal{B}(C)} \times \delta_C \rightarrow 2^{\mathcal{B}(C)}$.*

Notice that the update function does not simply "select" behaviors that abide to a given decision, rather it computes the consequence of a decision as a function of the current possible behaviors of the component.

For instance, suppose again we have a state variable component that models a light bulb with values $\{\mathrm{on}, \mathrm{off}, \mathrm{burned}\}$, and that no decision has been taken on the component. If a decision stating that the component should be turned on in a certain time interval is taken, the update function should be implemented to "limit" the behaviors to only those combinations for which the component does not take the value off anywhere in the given time interval. The f^U function for the light bulb component implements the fact that a light bulb can be either on or burned as a result of the decision to turn it on, but certainly not off. Furthermore, notice that the update function can also prescribe an increase in the possible combinations of behaviors: for instance, a decision to replenish a consumable resource will not limit but add possible behaviors from the time instant it is applied.

The update function thus implements how a component can react to decisions taken on it. For instance, the effect of a decision could equate to "keep all the behaviors that are equal to d' in t_1", or to "all the behaviors must be set to d'' after t_2". Given a decision on a component with a given set of behaviors, the update function computes the resulting set as in Figure 4.

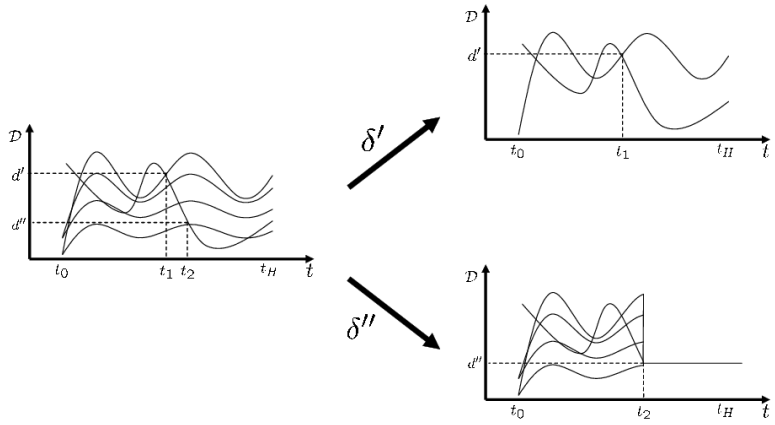


Figure 4: The update function computes the results of a decision on a component's set of behaviors. The figure exemplifies this effect given the two decisions: δ' imposes a value d' for the behaviors of the component in the time instant t_1; δ'' imposes that the values of all behaviors converge to d'' after time instant t_2.

In the following we present some further examples of decisions for the different

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 cut out 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 summing the value required by this allocation in the given interval. Outside the specified interval the profile is not affected.

Resource (Consumable). *Temporal element:* a TI. *Value:* quantity of resource produced or consumed at the given instant — a decision is either a *production* or a *consumption* of resource that arises in a time instant (like described in [31]). *Update function:* the resource profile is modified by summing (or subtracting) this production (or consumption) from the time it occurs on. Before the specified time instant, the profile is not affected.

It is also important to notice that the effect of a decision (i.e., the result of the update function) does not necessarily "restrict" the component's set of behaviors. For instance, given a consumable resource component as described in the above example, while a *consumption* decision will inevitably reduce the set of admissible behaviors for the component (all resource profiles which over-consume the resource with respect to the consumption which has been decided in the given temporal element have to be discarded), a *production* decision (i.e., a replenishment of the resource) entails an enlargement of the set of behaviors (the increased availability of the resource enlarges the set of admissible resource usage profiles).

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.

3.2 Interaction Among Components: the Domain Theory

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 inter-component relations in what we call a *domain theory*. As the consistency function describes which behaviors are consistent for each single component with their own "internal physical model", the domain theory defines which combinations of behaviors of all components are acceptable with respect to decisions taken on other components. We represent such requirements by means of *synchronizations*.

Definition 5 *Given n components $\mathbb{C} = \{C_1, \ldots, C_n\}$, a* domain theory *is a collection of* synchronizations. *A synchronization is 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 the* target *component). A synchronization has the form*

$$\langle C, d \rangle \longrightarrow \langle \mathbb{C}' = \{C'_1, \ldots, C'_n\}, \left\{ d'_1, \ldots, d'_{|\mathbb{C}'|} \right\}, \mathcal{R} \rangle$$

where: C is the reference component; d is the value of a component decision on C which makes the synchronization applicable; $\mathbb{C}' \subseteq \mathbb{C}$ is a set of target components on which new decisions with values d'_j will be imposed; and $\mathcal{R}$ is a set of relations *which bind the reference and target decisions.*

In order to clarify how such inter-component relationships are modeled as a domain theory, we give an instance (see the boxed example), showing a formalization of a well-known domain introduced by Allen in [2].

In the example we have implicitly introduced the concept of *relation* among component decisions. Specifically, the synchronizations all prescribe that the values d and d' in the reference and target components must occur together in time (DURING relation).

In general, OMPS provides two types of relations: *temporal* and *value* relations. In addition, *parameter* relations are provided if decision values $d(x_i, \ldots, x_n)$ depend on parameters.

Definition 6 *Given two component decisions $\delta_i = \langle d_i, \tau_i \rangle$ and $\delta_j = \langle d_j, \tau_j \rangle$,*

- *a* temporal relation *is a constraint among the temporal elements τ_i and τ_j of the two decisions δ_i and δ_j;*
- *a* value relation *is a constraint among the values d_i and d_j of the two decisions δ_i and δ_j;*
- *a* parameter relation *is a constraint among the parameters of the values d_i and d_j of two decisions δ_i and δ_j.*

A temporal relation among two decisions A and B can prescribe both qualitative and quantitative temporal requirements (e.g., A EQUALS B, or A OVERLAPS [min,max] B) thus using an hybrid of Allen's interval algebra [1] and quantitative temporal expressions [15].

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, parameter 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., ?bitrate $\leq 0.5 \cdot$?bandwidth.

Example. The problem describes the door of the Computer Science Building at Rochester. Because of its peculiar design, opening it requires two hands. In fact, a spring lock must be held up with one hand, while the door handle is held down with the other hand. This domain requires synchronization between two actions: the act of opening the door requires pulling down the handle *while* the spring lock is held up (we assume that pushing the door is not an issue, thus opening the door does not require additional action other than holding up the lock and holding down the handle). This problem can be formulated in terms of time and resource constraints. Notice that explicit concurrency is a problem for classical planners, thus this example, albeit simple, is quite challenging. For the problems involving classical planners and an alternative solution with respect to the present one, the interested reader is referred to [2].
This simple domain can be modeled in OMPS using five components: three state variables describing the physical environment with which interaction occurs: (DOOR, HANDLE and SPRING_LOCK) and two binary resources describing the hands on an executing agent (LEFT_HAND and RIGHT_HAND). These state variables and resources are conceptually binary: the DOOR component's behaviors are functions of time with values in $\{$*Open()*, *Shut()*$\}$, the hands in $\{0, 1\}$, etc.
In OMPS the goal of opening the door is seen as a decision stating that state variable DOOR must take the value *Open()* in a certain time interval. In our domain theory we can specify that during the interval in which the component DOOR takes the value *Open()*, the components HANDLE and SPRING_LOCK must take the value *Held_Down()* and *Held_Up()* respectively. Thus the synchronization among the DOOR, HANDLE and SPRING_LOCK components in the domain is:

$$\langle \text{DOOR}, \textit{Open()} \rangle \rightarrow \langle \{\text{HANDLE}, \text{SPRING\_LOCK}\}, \{\textit{Held\_Down()}, \textit{Held\_Up()}\}, \{\text{DURING}, \text{DURING}\} \rangle$$

expressing the fact that the decision to hold down the handle and to hold up (unlock) the spring lock must be taken concurrently DURING door opening. Moreover, both actions to hold the handle down and the spring lock up require using hands: this can be stated in OMPS terms by specifying the two sets of synchronizations:

$$\langle \text{HANDLE}, \textit{Held\_Down()} \rangle \rightarrow \langle \text{LEFT\_HAND}, 1, \{\text{DURING}\} \rangle$$
$$\langle \text{HANDLE}, \textit{Held\_Down()} \rangle \rightarrow \langle \text{RIGHT\_HAND}, 1, \{\text{DURING}\} \rangle$$

and

$$\langle \text{SPRING\_LOCK}, \textit{Held\_Up()} \rangle \rightarrow \langle \text{LEFT\_HAND}, 1, \{\text{DURING}\} \rangle$$
$$\langle \text{SPRING\_LOCK}, \textit{Held\_Up()} \rangle \rightarrow \langle \text{RIGHT\_HAND}, 1, \{\text{DURING}\} \rangle$$

stating, respectively, that either the left or the right hand must be used to manipulate the handle and another to manipulate the spring lock. Notice that the impossibility of using the same hand to hold up the spring lock and to hold down the handle concurrently is guaranteed by the consistency function of the hand components, which states that these resources cannot be used beyond their capacity.

4 Timeline-Driven Problem Solving

In the previous section we have introduced the fundamental building blocks of our particular approach to timeline-based planning. Planning in OMPS occurs by posting decisions and relations on a decision network which is maintained throughout the solving process and from which one or more solutions to the planning problem are extracted.

Definition 7 *Given a set of components $\mathbb{C} = \{C_1, \ldots, C_n\}$, a* decision network *is a graph $\langle V, E \rangle$, where each vertex $\delta_C \in V$ is a component decision defined on a component $C \in \mathbb{C}$, and each edge $(\delta^m_{C^i}, \delta^n_{C^j})$ is a temporal, value or parameter relation among component decisions $\delta^m_{C^i}$ and $\delta^n_{C^j}$.*

During planning, component decisions are inserted into the decision network, and relations among these decisions are imposed and propagated. As mentioned previously, the decisions in the decision network represent a flexible allocation of the decisions in the space of time, values and parameters. In order to maintain such flexibility in time, every component decision's temporal element (which can be a TI or TIN) is maintained in an underlying *flexible temporal network* as a Simple Temporal Problem (STP) [15]. An STP is a constraint-based representation where variables represent time points in a temporal interval between an *origin* O and an *horizon* H and binary constraints establish a minimal and maximal temporal distance among those time points. More in detail, an STP is a CSP were each time point is a variable defined over a continuous domain $[O, H]$ ($O, H \in \mathbb{R}, H > O$). Constraints between the time points are binary, and specify the allowed distance between the involved time points. Formally, given two time points t_i and t_j, a constraint between them in an STP is an interval $[min, max]$ such that $min \leq t_j - t_i \leq max$. Every time a constraint is added or removed, a propagation procedure is called to recalculate lower and upper bounds for each time point. The procedure will fail if the domain of one or more variables becomes empty. In this case the temporal problem is *inconsistent*, i.e., posted constraints are contradictory in what they require about the distances between time points. The solution obtained by selecting for each time point its lower bound value is called the *earliest start time solution*.

As a partially ordered plan contains one or more totally ordered plans in action-based planning, or a fully propagated constraint network contains one or more solutions for a CSP, a decision network contains a number of timelines. This is a consequence of maintaining the temporal elements of all decisions in an STP. More precisely, a timeline in OMPS is defined as follows:

Definition 8 *A* timeline *is an ordered sequence of component values. This sequence is determined 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 5 shows an example decision network consisting of four decisions $\delta^1_{C_1}$, $\delta^2_{C_1}$, $\delta^1_{C_2}$ and $\delta^2_{C_2}$ taken on two components C_1 and C_2, where the co-domain $\mathcal{D}_1$ of C_1's behaviors contains the values $A(x)$, $B(x)$ and $C(x)$ while the co-domain $\mathcal{D}_2$ of C_2's behaviors contains the values $C(x)$ and $D(x)$ (where x is an integer parameter). At the bottom of Figure 5 two of the timelines that can be extracted from the decision network are shown. Specifically, the figure shows the Earliest Start Time (EST) timeline and the Latest Start Time (LST) timeline. The former is obtained by instantiating all the start times of the decisions to the earliest point in time allowed by the temporal relations in the decision network, while the latter is obtained by allocating them to their latest start time. In both timelines shown in the example the allocation of parameters, values as well as the temporal allocation of the decisions is such that no relation in the decision network is violated: the parameters are instantiated so as to satisfy the parameter relations $x_6 = 3$ and $x_7 = 2x_6$ (in this example for simplicity, also x_1, x_2 and x_5 are assigned values); the value EQUALS relation between $\delta^2_{C_1}$ and $\delta^2_{C_2}$ prescribes that the two decisions should impose the same value, thus $\delta^2_{C_1}$ and $\delta^2_{C_2}$ can only take on

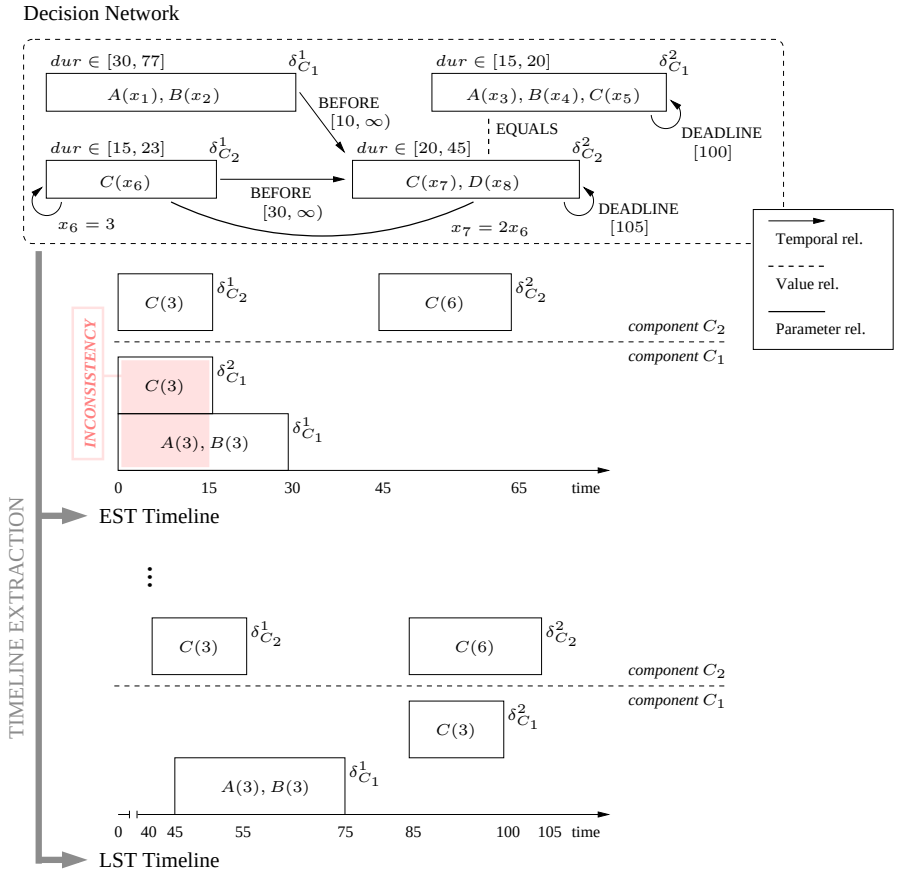


Figure 5: A decision network and two of the timelines that can be extracted from it (EST = earliest start time, LST = latest start time).

the common value value $C(\cdot)$ (i.e., the intersection of the two sets of possible values $\{A(\cdot), B(\cdot), C(\cdot)\}$ and $\{C(\cdot), D(\cdot)\}$); and so on.

Notice that a number of different timelines can be obtained from the decision network, each of which having a different allocation of the decisions in time lying "between" the EST and LST.

Finally, notice also that there is no relation in the decision network that disallows the decisions $\delta^1_{C_1}$ and $\delta^2_{C_1}$ to overlap. However, due to the value EQUALS relation between $\delta^2_{C_1}$ and $\delta^2_{C_2}$, $\delta^2_{C_1}$ must take on value $C(\cdot)$. This implies that in the EST timeline the decisions $\delta^1_{C_1}$ and $\delta^2_{C_1}$ cannot take on the only common values $A(\cdot)$ and $B(\cdot)$ between them, which represents a contradiction, as the two decisions overlap in time (in the interval $[0, 15]$). This is what we call an inconsistency. Notice also that although the inconsistency does not occur in the LST timeline (the only two overlapping decisions are related to different state variables), there is no guarantee as to whether this occurs in other timelines that lie between the EST and LST timelines. As we will see,

one of the roles of the planner is precisely to modify the decision network (by adding relations) so as guarantee inconsistency-free timelines.

4.1 Timeline Management: Resolving Gaps and Inconsistencies

In the current implementation, we follow for every type of component an earliest start-time approach, i.e., we extract a timeline where all component decisions are assumed to occur at their earliest start time and last the shortest time possible.

Figure 6 shows the EST timeline corresponding to a decision network in which three decisions have been taken on one component of type state variable. The example illustrates two properties of timelines, namely *gaps* and *inconsistencies*.

Gaps. The first of these features depends on the fact that decisions imposed on the state variable do not result in a complete coverage of the planning horizon with decisions (as in the interval $[30, 40]$ in the figure). A gap 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 process of deciding which value(s) are admissible with respect to the state variable's internal consistency function (i.e., the component's f^C function) constitutes the process of *timeline completion* — timeline completion is carried out by the planner in order to modify the plan to avoid gaps in the timeline.

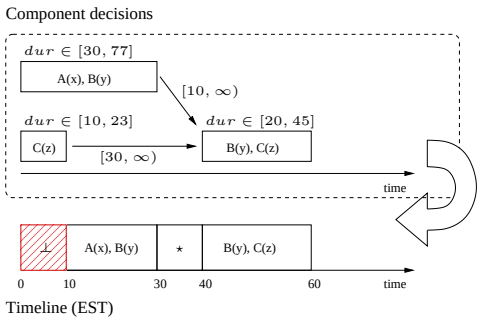


Figure 6: Three decisions on a state variable, and the resulting earliest start time (EST) timeline.

Notice that timeline completion is required for components such as state variables, where not all intervals of time are necessarily covered by a decision. Conversely, reusable resources as we have defined them in this paper do not need a timeline completion step since an interval of time in which no decision is taken (a gap) implies that the resource is simply unused (thus it is not a condition that should be subject to planning). When invoked, timeline completion adds new decisions to the decision network.

Inconsistencies. In addition to gaps, inconsistencies can arise in the timeline, as is the case in the interval $[0, 10]$ in Figure 6. The nature of inconsistencies depends on the specific component we are dealing with. In the case of state variables, an inconsistency occurs when two or more decisions whose intersection is empty overlap in time. As opposed to gaps, inconsistencies do not require the generation of additional component decisions, rather they can be resolved by posting further temporal constraints on the plan. For instance, two decisions $((A(x), B(y)), \tau_1)$ and $(C(z), \tau_2)$ whose temporal elements τ_1 and τ_2 can overlap in time lead to an inconsistency which can be resolved

by imposing a temporal relation which forces τ_2 to occur either after or before τ_1. In the case of a resource component, an inconsistency occurs when the activities in the decision network require more resources than are available in a temporal interval. In general, we call the process of resolving inconsistencies *timeline scheduling* — the scheduling process deals with the problem of removing inconsistencies in the timeline.

As mentioned, the process depends on the component. For a resource, activity overlapping results in an inconsistency if the combined usage of the overlapping activities requires more than the resource's capacity. For a state variable, any overlapping decisions that require a conflicting set of values constitute an inconsistency (as in the example in Figure 5).

4.2 The OMPS Solving Process

A fundamental difference between action-based planning and the OMPS approach lies in the notion of support. In action-based planning, actions can be applied if and only if they are supported in the previous state of the state space. Asserting that an action is supported in action-based planning equates to verifying that all of its preconditions are satisfied. This notion is essentially limited to causal support, as preconditions are predicates that can be true or false. We find this in OMPS as well: a decision which is obtained as the instantiation of the reference of a synchronization $\langle C, d\rangle \longrightarrow \langle \mathbb{C}', \left\{d'_1, \ldots, d'_{|\mathbb{C}'|}\right\}, \mathcal{R}\rangle$ "supports" another decision which asserts value d'_j on component C_j while satisfying the relations in $\mathcal{R}$.

The notion of support in action-based planning corresponds to the notion of support in OMPS in the case that (1) $\mathbb{C}' = \{C\}$ and (2) $\mathcal{R}$ contains only the temporal relation AFTER $[0, +\infty]$. In OMPS decisions are added to the decision network when support is found for them. This process of adding decisions when they are supported is called (similarly to action-based planning) *subgoaling*. When a decision already present in the network has been used for subgoaling, we say that this decision is *justified*.

Clearly, OMPS offers a great deal of flexibility: not only can relations represent more complex temporal constraints, but they can also represent quantitative parameter and value relations. Indeed in OMPS we generalize the concept of support to include all three dimensions. In action-based planning actions are supported only as a consequence of applying the domain theory (or if they represent initial conditions). In OMPS on the other hand, decisions are supported by means of applying the synchronizations in the domain (i.e., subgoaling). To continue the comparison, the process of planning in the former approach consists in iteratively synthesizing actions that are supported in the current state (or states) in the search space towards the aim of obtaining the (propositional) goals; in OMPS, planning consist in justifying decisions. This occurs either by supporting new decisions (i.e., subgoaling) or by unifying unjustified decisions with others that are. The mechanisms of justification and unification are explained below.

The input to the OMPS planning process consist of a decision network with unjustified decisions and/or that leads to timelines with gaps or inconsistencies to resolve. Indeed, the initial decision network contains both "initial conditions" and "goals" as they are defined in action-based planning. Initial conditions are decisions that do not lead to the process of subgoaling. In other words, initial conditions describe parts of

timeline that the planner cannot change and that do not require synchronization. Conversely, goals directly or indirectly entail the need to apply synchronizations in order to reach domain theory compliance. In other words, a goal consists of a set of unjustified decisions which are intended to trigger the solving strategy to exploit the domain theory's synchronizations to synthesize decisions.

The output of the solving process is a set of timelines, one for each component in the domain, that are fully consistent and without gaps. More specifically, the overall solving process can be detailed as follows (see the algorithm on the next page).

The first step in the planning process is domain theory application, whose aim is to attempt to justify all decisions in the decision network. If there is no way to justify all decisions in the plan, the algorithm fails (lines 2-3).

Once every decision has been justified, the solver tries to extract a timeline for each component (line 4, where T_{EST} is the set of all the Earliest Start Time timelines extracted). At this point, it can happen that some timelines are not consistent, meaning that there exists a time interval over which conflicting decisions overlap (an inconsistency). In such a situation, a scheduling step is triggered (lines 5-8). If the scheduler cannot solve all conflicts or generated scheduling constraints fail to propagate (with respect to the other temporal, value and parameter constraints) when added to the decision network ($propagate$ procedure), the solver backtracks directly to supporting decisions (i.e., searching for different ways of applying the synchronizations in the domain theory to justify decisions). It is worth pointing out that this algorithm aims to introduce a general framework for planning with timelines, and it is parametric with respect to both the timeline extraction procedure and the scheduling algorithm applied to remove inconsistencies. For instance, our current implementation applies a simple EST timeline extraction procedure previously introduced, but other more complex procedures can be envisaged.

If the solver manages to extract a conflict-free set of timelines (one for each component), it then triggers a timeline-completion step (lines 10-13) on any timeline which is found to have gaps (state variables for instance). It may happen that some timelines cannot be completed or the produced decisions and constraints fail to propagate when added to the decision network. In this case, the solver backtracks again to the previous decision supporting step and again searches for another way of justifying all decisions. If the completion step succeeds for all timelines, the solver returns to domain theory application, as timeline completion has added decisions which are not justified.

Once all timelines are consistent and complete, the solver is ready to extract behaviors (lines 15-19). If behavior extraction fails, the solver attempts to backtrack to timeline completion (in case of behavior extraction failure we try first different completions instead of backtracking directly to the support decision step to change the plan). Finally, the whole process ends when the solver succeeds in extracting at least one behavior for each timeline. This collection of mutually consistent behaviors represents a fully instantiated solution to the planning problem.

In summary, the solving process underlying our approach is fundamentally an iterative procedure: it observes the current decision network (which is essentially the initial condition at iteration zero) and attempts to justify the presence of all decisions

Function `solve`(*Components* $\mathbb{C}$, *DecisionNetwork* $\langle V, E\rangle$,*DomainTheory* Th)

```
1  while ∃ unjustified decision ∈ V do
2      if ¬supportDecisions(⟨V, E⟩, Th) then
3          return false;
4      T_EST ← extractTimeline(ℂ);
5      while ∃ inconsistency in T_EST do
6          E′ ← scheduleTimeline(T_EST);
7          if ¬propagate(⟨V, E ∪ E′⟩) then
8              backtrack to supportDecisions;
9      T_EST ← extractTimeline(ℂ);
10     while ∃ gap to fill in T_EST do
11         ⟨V′, E′⟩ ← completeTimeline(T_EST);
12         if ¬propagate(⟨V ∪ V′, E ∪ E′⟩) then
13             backtrack to supportDecisions;
14 T_EST ← extractTimeline(ℂ);
15 Σ ← extractBehavior(T_EST);
16 if Σ ≠ null then
17     return true;
18 else
19     backtrack to completeTimeline;
```

— where a decision can be justified because it supports other decisions (through a synchronization with another component) or because it unifies with another decision which is already present and justified; the solving process then turns to each component's timeline to see the results of the previous step, and isolates inconsistencies and gaps — this is the core of the solving process, as timelines provide a "time-instantiated view" of the current decision network; lastly, if timeline completion has lead to the introduction of new decisions (which are not justified), the iteration starts over.

Once this iterative process has reached a point where all decisions are justified and there exists a timeline for each component that has no gaps or inconsistencies, a solution is extracted, i.e., all decisions are fully instantiated in time, a value is chosen among all possible values for each decision, and each chosen value is instantiated with respect to its parameters. If this step terminates successfully, the set of all obtained behaviors represents a solution, otherwise the planner must backtrack.

Notice that scheduling is subsumed in the timeline management step. Indeed, scheduling is employed to resolve inconsistencies, and feeds back into the solving iteration through additional temporal constraints. Scheduling is fundamentally interleaved with the more "planning-oriented" steps (domain theory application and timeline completion), thus making OMPS an example of strongly integrated planning and scheduling.

In the following, we describe the principal elements of the $supportDecision()$,

$extractTimeline()$ and $extractBehavior()$ procedures employed in the above solving loop.

Supporting decisions. The process of supporting decisions consists in iteratively tagging decisions as justified in two different ways (see the algorithm on the next page):

1. **Unification.** A decision can be *unified with another decision* in the network if a temporal EQUALS and a value EQUALS constraint can be posted between the two decisions. If δ_C *unifies* with another decision in the network, then δ_C can be marked as justified and the unifying relations (generated by means of the $unify$ procedure) are added to the decision network (lines 9–13).

2. **Expansion.** A decision $\delta_{C_i} = \langle d_i, \tau_i \rangle$ *unifies with the reference value* $\langle C, d \rangle$ *of a synchronization* if $C_i = C$ and d_i can be made equal to d[2] (we denote this possibility with $d_i \approx d$). If δ_{C_i}'s value *unifies* with the reference value of a synchronization in the domain theory, then δ_{C_i} can be marked as justified provided that (lines 15–25) target decision(s) generated from synchronization's required values and relations (by means of the $inst$ procedure) are added to the decision network (subgoaling) and propagated (as not justified, thus subject to the next invocation of supportDecisions()). In case there is no synchronization that can be applied, the decision is directly marked as justified, thus not leading to subgoaling. If an expansion step fails (because the propagations triggered by subgoaling fail), before failing the whole procedure we try to backtrack choosing a different synchronization (whenever available) to justify the decision.

Extracting timelines. In order to obtain a total ordering among the "floating" decisions in the decision network, a timeline must be extracted from the decision network. Specifically, this process depends on the component for which extraction is performed. For a resource, for instance, the timeline is computed by ordering the allocated activities and summing the requirements of those that overlap. For a state variable, the effects of temporally overlapping decision are computed by intersecting the required values, to obtain (if possible) in each time interval a value which complies with all the decisions that overlap during the time interval.

Extracting behaviors. Once the OMPS solving process has successfully converged on a set of timelines with no inconsistencies or gaps, one or more consistent behaviors can be extracted from each component's timeline. A behavior is one particular choice of values for each temporal segment in a component's timeline. Each behavior that can be extracted from a component's timeline represents, together with a behavior for every other component, a single unique solution to the planning problem. The OMPS solving process has filtered out all behaviors that are not consistent with respect to the domain theory and the components' consistency function. Nevertheless, herein lies another problem. Although we are guaranteed that a selection of behaviors exists for each component, the solving process has not thus far provided us with the specific

[2]That may involve the generation of value and/or parameter constraints on δ_{C_i}.

Function `supportDecisions(`*DecisionNetwork* $\langle V, E\rangle$*,DomainTheory Th*`)`

1 $S \leftarrow \{\delta \in V : \delta$ not justified $\}$;
2 **while** $S \neq \emptyset$ **do**
3 select $\delta_C = \langle d, \tau\rangle \in S$;
4 $U_{\delta_C} \leftarrow \{\delta \in V \backslash S : \delta$ unifies with $\delta_C\}$;
5 **if** $U_{\delta_C} \neq \emptyset$ **then**
6 select strategy $\in$ {unify, expand};
7 **else**
8 strategy = expand;
9 **if** *strategy = unify* **then**
10 select $\delta_U \in U_{\delta_C}$;
11 mark δ_C as justified;
12 $S = S \backslash \{\delta_C\}$;
13 $E = E \cup \{unify(\delta_C, \delta_U)\}$;
14 **else**
15 **if** $f^U(2^{\mathcal{B}(C)}, \delta_C) \neq \emptyset$ **then**
16 mark δ_C as justified;
17 $S = S \backslash \{\delta_C\}$;
18 $E_{\delta_C} \leftarrow \left\{\langle C, d\rangle \longrightarrow \langle \mathbb{C}', \left\{d'_j\right\}, \mathcal{R}\rangle \in Th : d_i \approx d\right\}$;
19 **if** $E_{\delta_C} \neq \emptyset$ **then**
20 select $\langle C, d\rangle \longrightarrow \langle \mathbb{C}', \left\{d'_1, \ldots, d'_{|\mathbb{C}'|}\right\}, \mathcal{R}\rangle \in E_{\delta_C}$;
21 $\langle V', E'\rangle \leftarrow inst(\langle C, d\rangle \rightarrow \langle \mathbb{C}', \left\{d'_1, \ldots, d'_{|\mathbb{C}'|}\right\}, \mathcal{R}\rangle, \delta_C)$;
22 **if** $\neg$propagate$(\langle V \cup V', E \cup E'\rangle$ **then**
23 backtrack to synchronization selection in E_{δ_C} (line 20);
24 **else**
25 **return** false;
26 **return** true;

combinations of behaviors that are together consistent with the domain theory. Specifically, since every segment of a timeline potentially represents a disjunction of values, solution extraction must choose specific behaviors consistently. This task is carried out in OMPS as a post-processing step, the details of which are outside the scope of this paper. Notice however that due to the least commitment nature of the overall OMPS planning process, the existence of a consistent selection of behaviors is guaranteed.

5 The OMPS Software System

In the previous sections we have introduced the concept of component-based approach in OMPS and we have shown an iterative process for manipulating timelines in order to solve a given planning and/or scheduling problem. We now briefly sketch the current

implementation of OMPS, initially describing the software architecture and thereafter presenting an example of use.

The OMPS software architecture consists of layers organized in a hierarchy. Each layer is responsible for dealing with a particular aspect of the planning and scheduling problem, and each layer uses the services provided by the underlying layers to implement its functionalities. The constraint-based nature of the approach is extremely visible in the way the different layers exchange information: constraints are posted on the underlying levels as a consequence of decisions taken on higher levels, and decisions are taken on higher levels by analyzing the domains of the variables in the underlying levels.

OMPS's architecture is subdivided into four levels: a *Temporal* layer, a *Component* layer, a *Domain* layer, and a *Solver* layer. Layers are organized according to the hierarchy shown in Figure 7.

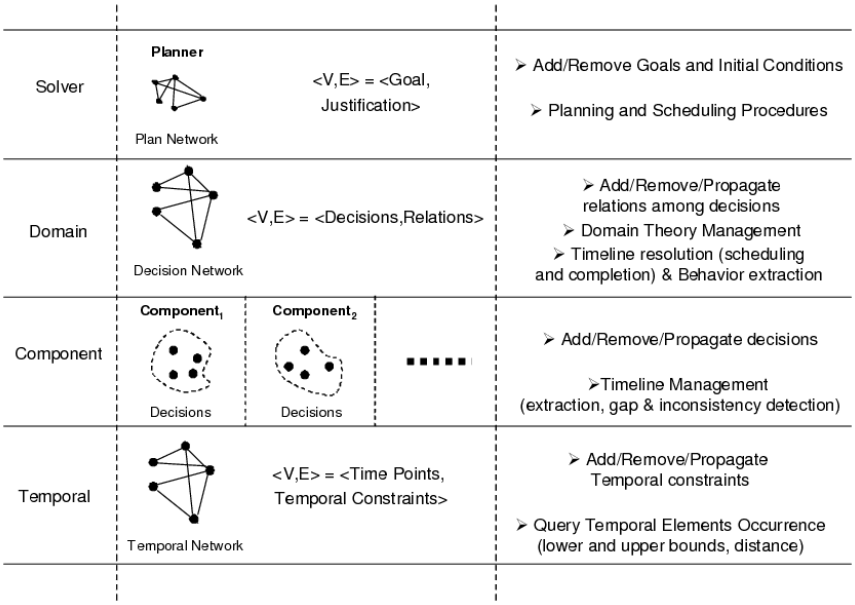


Figure 7: The layered implementation of the OMPS system.

The **Temporal** layer manages temporal information. It maintains a temporal network to compute the effects of *temporal constraints* over a set of *time points*. It lies at the bottom of the hierarchy and provides to higher levels primitives to manage time instants and time intervals. In the current implementation of OMPS, the chosen temporal model is the STP [15].

The **Component** layer is the point of expansion of the OMPS architecture. In this architecture a component is a module able to compute the effects of *decisions* (gener-

ated by higher levels) over its *behaviors*. A component provides to higher levels basic timeline-management primitives (like timeline extraction, gap- and inconsistency-detection). It is a point of expansion because the components make the architecture independent from the actual implementation of the functionalities they provide, encapsulating component-specific algorithms and hiding differences about behaviors, inconsistency detection and resolution behind a common interface. For instance in the current implementation reusable resources implement propagation algorithms from [31, 5] for consistency detection and timeline extraction, while state variables use algorithms from [23].

The **Domain** layer manages *relations* among decisions maintaining the *decision network*. This is the level where concurrent threads represented by each component in the underlying level are put together to constitute the component-based domain: this level is in fact responsible for providing all domain theory management functions (like the computation of expansion and/or unification possibilities) to generate synchronizations among components and to allow the resolution of higher level problem solving (e.g., decision justification). The domain level also provides complete timeline management functionalities (like timeline completion and conflict resolution) and behavior extraction utilities.

The **Solver** level maintains a data structure, the *plan network*, that is a hypergraph where the nodes are goals, which in turn are either decisions or relations among decisions, and the edges are *justifications* for the presence in the plan of the goals they connect. In this level the algorithms presented in Section 4 are implemented, exploiting functionalities provided by lower levels. In the current implementation there are four kinds of edges in the plan network: a supporting decision edge (subdivided into a domain theory expansion edge and a unification edge), a timeline scheduling edge, a timeline completion edge and a behavior extraction edge. The plan network constitutes the planner and scheduler search space.

It is worth recalling here that the architecture has been conceived to be easily extensible by adding components. This capability is very important to achieve a good balance between general, domain independent planning (easily customizable to various domains) and specialized, efficient reasoning (often needed in real world domains for efficiency reasons). In the next subsection we present an example of modeling and planning with OMPS, in a space related domain: in this example this architecture will be extended by plugging in a new component, the battery, needed to model a non trivial behavior in this domain.

5.1 Using OMPS

We present here an example inspired from current practice at the European Space Agency (ESA) mission control center.

The planning problem consists in deciding data transmission commands from a satellite orbiting Mars to Earth within given ground station visibility windows. The spacecraft's orbits for the entire mission are given, and are not subject to planning.

The fundamental elements which constitute the system are: the satellite's Transmission System (TS), which can be either in "transmit mode" on a given ground station or idle; the satellite's Pointing System (PS); and the satellite's battery (BAT). In addition, an external, uncontrollable set of properties is also given, namely Ground Station Visibility (GSV) and Solar Flux (SF). Station visibility windows are intervals of time in which given ground stations are available for transmission, while the solar flux represents the amount of power generated by the solar panels given the spacecraft's orbit. Since the orbits are given for the entire mission, the power provided by the solar flux is a given function of time $\mathrm{sf}(t)$. The satellite's battery accumulates power through the solar flux and is discharged every time the satellite is slewing or transmitting data. Finally, it is required that the spacecraft's battery is never discharged beyond a given minimum power level (in order to always maintain a minimum level of charge in case an emergency manoeuvre needs to be performed).

A domain theory instantiating this example in OMPS is provided in the appendix at the end of the paper, where we briefly introduce the DDL.3 Domain Definition Language designed for OMPS. Specifically, we define the following five components:

PS, TS and GSV. The spacecraft's pointing and transmission systems (PS and TS), as well as ground station visibility (GSV)are modeled with three state variables. The consistency functions of these state variables (possible states, bounds on their duration, and allowed transitions) are depicted in Figure 8. They are expressed in the OMPS formalism in the definition of the component *types*, lines 11–43 in the DDL.3 domain in the appendix. Synchronizations in the OMPS formalism are expressed on *instantiations* of the component types. Specifically, lines 56–58 instantiate the three state variables (`Pointing_System`, `Transmission_System` and `Ground_Station_Vis`). The synchronizations modeled for the state variables are shown in Figure 8: one states that the value "locked(?st3)" on component PS requires the value "visible(?st6)" on component GSV during its temporal occurrence in the timeline (where ?st3 = ?st6, i.e., the two values must refer to the same station, lines 64–67); another synchronization asserts that transmitting on a certain station requires the PS component to be locked on that station during its temporal occurrence (lines 79–80); lastly, both slewing and transmission entail the use of a constant amount of power from the battery exactly in the same temporal interval in which the operation causing the consumption occurs (lines 69–72 and 81–83).

SF. The solar flux (SF) is modeled as an input function for the battery component. Given that the flight dynamics of the spacecraft are given (i.e., the angle of incidence of the Sun's radiation with the solar panels is given), the profile of the solar flux component is a given function of time $\mathrm{sf}(t)$ which is not subject to changes. This function is provided in a file by ESA and it is ingested into the solving process directly as a timeline of the `Solar_Panel_Charge` component type (line 50). The `Solar_Flux` component is an instantiation of this type (line 60). Notice that decisions are never imposed on this component (i.e., the SF component has only one behavior), rather its behavior is solely responsible for determining power production on the battery.

BAT. The spacecraft's battery component is modeled as follows. Its consistency func-

tion consists in a maximum and minimum power level ($\max, \min$), the former representing the battery's maximum capacity, the latter representing the battery's minimum depth of discharge. The BAT component's behavior is a temporal function $\mathrm{bat}(t)$ representing the battery's level of charge. Assuming that power consumption decisions resulting from the TS and PS components are described by the function $\mathrm{cons}(t)$, the update function calculates the consequences of power production ($\mathrm{sf}(t)$) and consumption on $\mathrm{bat}(t)$ as follows:

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

where L_0 is the initial charge of the battery at the beginning of the planning horizon and α is a constant parameter which approximates the charging profile. The BAT component is modeled in the OMPS formalism similarly to the SF component (see lines 48–49 and 59).

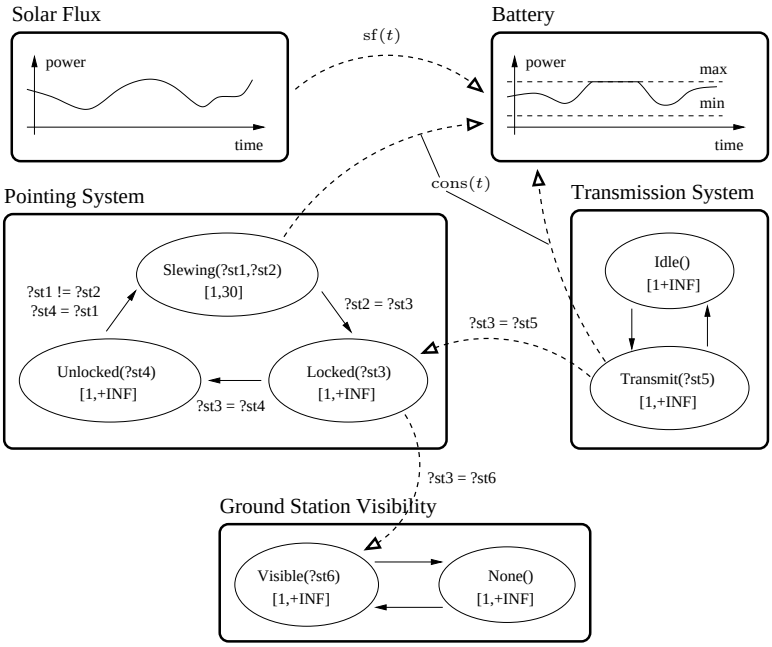


Figure 8: State variables and domain theory for the running example.

Figure 9 shows the timelines of the GSV and TS components resulting from the application of a set of initial condition and goals (nothing is specified for the PS component). Notice that the GSV timeline (in the lower half of the figure) is fully defined, reflecting the fact that the GSV component is not controllable, rather it represents the evolution in time of station visibility given the fully defined flight dynamics of the

satellite.

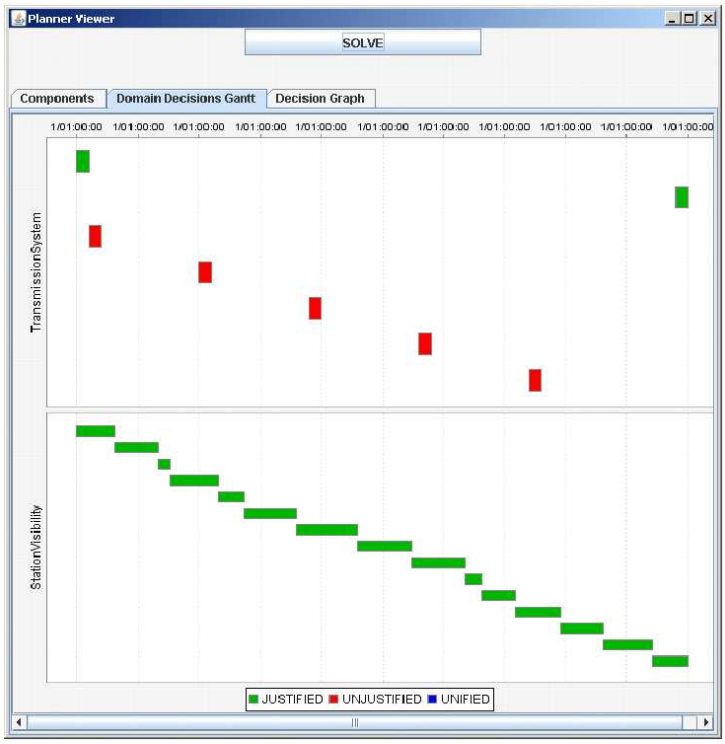


Figure 9: A set of initial conditions and goals shown in the initial timelines for the TS (top) and GSV (bottom) components.

The TS timeline (in the upper half in figure) contains five "transmit" value choices, through which we represent our goal (in the middle of the timeline) and two "idle" value choices that represents the initial conditions (the initial and final value for TS's timeline on the sides). These value choices are allocated within flexible time bounds, the planning process is in charge of defining their actual temporal positioning. As opposed to the GSV timeline, the TS timeline contains gaps, and it is precisely these gaps that will be "filled" by the solving algorithm. In addition, the application during the solving process of the synchronization between the GSV and PS components will determine the construction of the PS's timeline (which is completely void of component decisions in the initial situation), reflecting the fact that it is necessary to point the satellite towards the visible target before initiating transmission. The solution found by OMPS is shown in Figure 10. The PS timeline shown on the upper part in figure has been completely planned, while the TS timeline in the middle has been filled and proper temporal positions and lengths have been chosen for the goals. Finally the GSV timeline (in the lower part of the figure) has not been changed by the planner and has been used only to synchronize other timelines (the values in the bottom of the timeline hilights which spots have been chosen to transmit).

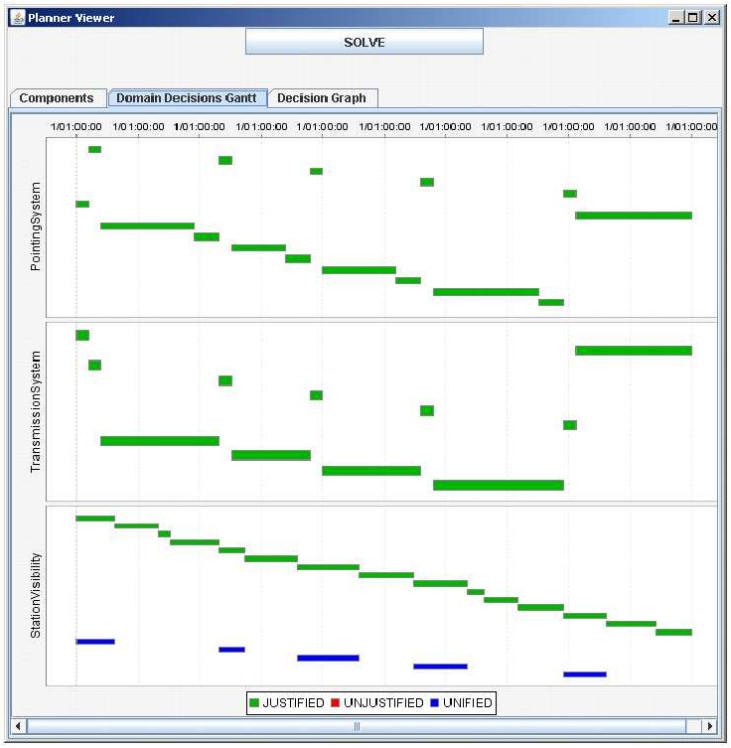


Figure 10: The PS (top), TS (middle) and GSV (bottom) timelines resulting from the OMPS solving process.

In the example above, we have employed components of three types: state variables to model the PS, TS and GSV elements, an external uncontrollable component to ingest the solar flux profile, and an ad-hoc component to model the spacecraft's battery. Notice that this latter component is essentially an extension of a reusable resource: whereas a reusable resource's update function is trivially the sum operator (imposing a decision on a reusable resource entails that the resource's availability is decreased by the value of the decision), the BAT's update function calculates the consequences of decisions as per the above integration over the planning horizon.

The ability to encapsulate potentially complex modules within OMPS components provides a strong added value in developing real-world planning systems. This brings with it a number of operational advantages, such as fast-prototyping, prototype reliability (a particularly important feature in deployed applications) and software scalability. Finally, the approach allows to leverage the efficiency of problem de-composition.

5.2 Connections with Similar Works

OMPS is a new addition to the family of timeline-based planning systems. Our work can be compared with the three planning architectures that share most of the features

introduced in this paper.

RAX-PS, EUROPA. RAX-PS [28], and its predecessor HSTS, have been the first to propose a modeling language with an explicit representation of state variables. OMPS extends significantly that architecture through the concept of component that generalizes what in RAX-PS is represented just with state-variables – e.g., see in [24] how resource reasoning is represented in terms of state variables and see [12] for an analysis of limitations of that representation. OMPS allows to explicitly reason on resource conflicts like *contention peaks* while in RAX-PS resources are seen as specialized state variables. While it is immediate to represent binary resources as state variables, integrating multi-capacity resources destroys the least commitment principle for aggregate or consumable resources. In both RAX-PS and OMPS quantitative temporal reasoning is the core of the constraint data-base. Conversely, the use of the decision network as a "Multi-CSP" in OMPS is a distinctive aspect of such system. It is also worth underscoring that the concept of causal constraints named *compatibilities* in RAX-PS, corresponds to the two concepts of *consistency functions* and *synchronizations* in OMPS. For models with state variables only, the functionality of the two systems is the same. However, OMPS adds the possibility to specify and reason upon synchronizations among components of different types (e.g., from a resource to a state variable and/or vice versa).

IxTeT. Also IxTeT [26] follows a domain representation philosophy based on state attributes which assume values on a domain. Indeed this architecture represents system dynamics with a STRIPS-like logical formalism. It follows more closely a CSP approach as a general frame of reference as done by OMPS. In contrast with our work, resources are called into play withing the action schema and their requirements are reasoned upon by a conflict analyzer on top of the plan representation. OMPS follows the approach of defining resources as separate components which may have integrated constraint propagation abilities like those described in [12, 31]. Resource requirements are described recurring to synchronization constraints in the OMPS domain theory. In general our choice allows an integration of state of the art resource propagation in a more native way.

ASPEN. The central data structure in JPL's Aspen architecture [14] is an activity representing an action either in a plan or in a schedule. Activities can use one or more resources, and have parameters whose values are instantiated by other activities. Activities are managed hierarchically in the ADB (Activity Data Base) which represents also resource usage constraints and some temporal constraints, including universally quantified conditions. Indeed, a separated Temporal Constraints Network exists, but the ADB does not represent the core feature of the architecture, which is instead built on local search algorithms for efficient problem resolution. The state variables (defined as an enumerated type or vector) play a minor role in this formalism: an activity may need a synchronization with some value (for each state variable we can specify only state changing rules). Resource timelines exist and are used for counting reservation amounts, but resource constraints are managed in the ADB. Indeed this architecture

does not focus much on flexible temporal plans as done in RAX-PS and OMPS but rather on efficiency of specialized search algorithms and in a planning process that involves linked activity instantiation and hierarchical expansion. As a general comment we may say that ASPEN is a remarkable effort as an engineered platform for different applications but is the most diverging architecture with respect to the CSP characterization we have introduced here.

6 Conclusions

In this paper we have presented a constraint-based approach for planning and scheduling. The basic underpinning of the approach is the concept of timeline, an instantiation in time of a network of decisions on components. Our approach unifies planning and schedulinginto a homogeneous solving framework. The result represents, in our view, a structured combination of various proposals presented by separated literature: on one hand, we employ ideas from other timeline-based planning architectures, such as HSTS [35, 34]; on the other, we integrate advanced constraint-based scheduling features [31, 11] within the very core of the planning process.

The proposed unified view on planning and scheduling is grounded on the concept of component, i.e., any set of properties that vary in time. This includes "classical" concepts such as state variables [35, 34, 10, 28, 21]), as well as renewable/consumable resources [11]. As we have shown, component-based domain modeling greatly facilitates the process of integrating ad-hoc reasoning functionalities within the solving framework: OMPS's component-oriented nature allows to modularize the reasoning algorithms that are specific to each type of component within the component itself, e.g., profile-based scheduling routines for resource inconsistency resolution are implemented within the resource component itself.

While the ability to schedule is common to many types of components, we have also given an example of more component-specific reasoning functionality. Specifically, the battery component essentially extends a reusable resource with the ability to maintain its power availability profile. The ability to encapsulate potentially complex modules within OMPS components provides a strong added value in developing real-world planning systems. This brings with it a number of operational advantages, such as fast-prototyping, prototype reliability and software scalability. Finally, the approach allows to leverage the efficiency of problem de-composition: scheduling occurs within the single components, and does not need to involve parts of the decision network which do not pertain the specific component which generates an inconsistency. Moreover (as shown for the battery component in the example), other specific reasoning functionalities can be teased out of the overall solving process and can be dealt with by the individual components when decisions are taken on them. OMPS is a framework for developing decision support tools, and the possibility to encapsulate functionalities in such a way contributes to producing lean and efficient software prototypes from the very first stages of development.

Acknowledgments. This work has been partially supported by the European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). APSI

partners are VEGA GmbH, ONERA, University of Milan and ISTC-CNR. Thanks to Angelo Oddi and Gabriella Cortellessa for their constant support, and to Carlo Matteo Scalzo for contributing an implementation of the battery component. The Authors wish to thank the anonymous reviewers for their in-depth and useful comments on an earlier version of this paper.

References

[1] James Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] J.F. Allen. Temporal Reasoning and Planning. In *Reasoning About Plans*. Morgan Kaufmann Pub., 1991.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[4] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling*, volume 39 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, 2001.

[5] Philippe Baptiste and Claude Le Pape. A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. In Chris S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 600–606. Morgan Kaufmann, August 1995.

[6] R. Bartak. Constraint Satisfaction for Planning and Scheduling. In I. Vlahavas and D. Vrakas, editors, *Intelligent Techniques for Planning*. Idea Group Publishing, 2004.

[7] J.C. Beck, A.J. Davenport, E.D. Davis, and M.S. Fox. The ODO Project: Towards a Unified Basis for Constraint-Directed Scheduling. *Journal of Scheduling*, 1:89–125, 1998.

[8] Peter Brucker. *Scheduling Algorithms*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[9] A. Cesta, S. Fratini, and A. Oddi. Planning with Concurrency, Time and Resources: A CSP-Based Approach. In I. Vlahavas and D. Vrakas, editors, *Intelligent Techniques for Planning*. Idea Group Publishing, 2004.

[10] A. Cesta and A. Oddi. DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains. In M. M.Ghallab and A. Milani, editors, *New Directions in AI Planning*. IOS Press, 1996.

[11] A. Cesta, A. Oddi, and S. F. Smith. A Constraint-based method for Project Scheduling with Time Windows. *Journal of Heuristics*, 8(1):109–136, January 2002.

[12] A. Cesta and C. Stella. A Time and Resource Problem for Planning Architectures. In *Proceedings of the Fourth European Conference on Planning (ECP 97)*, 1997.

[13] Cheng-Chung Cheng and Stephen F. Smith. Generating feasible schedules under complex metric constraints. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1086–1091, Seattle, Washington, USA, August 1994. AAAI Press/MIT Press.

[14] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. Fisher, T. Barrett, G. Stebbins, and D. Tran. ASPEN - Automated Planning and Scheduling for Space Mission Operations. In *Proceedings of SpaceOps 2000*, 2000.

[15] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, January 1991.

[16] Rina Dechter. *Constraint Processing*. Morgan Kaufmann Pub. Inc., 2003.

[17] Minh Binh Do and Subbarao Kambhampati. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, 132(2):151–182, 2001.

[18] Amin El-Kholy and Barry Richards. Temporal and resource reasoning in planning: The parcPLAN approach. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pages 614–618. Wiley & Sons, 1996.

[19] R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, 1971.

[20] M. S. Fox. Constraint Guided Scheduling: A Short History of Scheduling Research at CMU. *Computers and Industry*, 14(1–3):79–88, 1990.

[21] J. Frank and A. Jónsson. Constraint-based attribute and interval planning. *Constraints*, 8(4):339–364, 2003.

[22] Jeremy Frank, Ari K. Jónsson, and Paul Morris. On reformulating planning as dynamic constraint satisfaction. *Lecture Notes in Computer Science*, 1864, 2000.

[23] S. Fratini, A. Cesta, and A. Oddi. Extending a Scheduler with Causal Reasoning: a CSP Approach. In *Proceedings of Workshop on Constraint Programming for Planning and Scheduling*, 2005.

[24] R.E. Frederking and N. Muscettola. Temporal Planning for Transportation Planning and Scheduling. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1992.

[25] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through Stochastic Local Search and Temporal Action Graphs. *JAIR*, 2003. Special issue on 3rd International Planning Competition, to appear.

[26] M. Ghallab and H. Laruelle. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on Artifial Intelligence Planning Scheduling Systems*. AAAI Press, 1994.

[27] D. Ghallab M., Nau and P. Traverso. *Automated Planning, Theory and Practice*. Morgan Kaufmann Publishers, 2004.

[28] A.K. Jonsson, P.H. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *Proceedings of the Fifth Int. Conf. on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 2000.

[29] Henry Kautz and Bart Selman. Blackbox: A new approach to the application of theorem proving to problem solving. In *AIPS98 Workshop on Planning as Combinatorial Search*, pages 58–60, 1998.

[30] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[31] P. Laborie. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and new Results. *Artificial Intelligence*, 143:151–188, 2003.

[32] Jonathan M. Lever and Barry Richards. parcPLAN: A planning architecture with parallel actions, resources and constraints. In *Proceedings of 9th International Symposium on Methodologies for Intelligent Systems*, LNCS 869, pages 213–222. Springer Verlag, 1994.

[33] S. Mittal and B. Falkenheimer. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 25–32, 1990.

[34] N. Muscettola. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., editor, *Intelligent Scheduling*. Morgan Kauffmann, 1994.

[35] N. Muscettola, S.F. Smith, A. Cesta, and D. D'Aloisi. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems*, 12(1):28–37, 1992.

[36] A. Oddi and S. F. Smith. Stochastic Procedures for Generating Feasible Schedules. In *Proceedings 14^{th} National Conference on Artificial Intelligence (AAAI-97)*, pages 308–314, 1997.

[37] K. M. Passino and P. J. Antsaklis. A system and control theoretic perspective on artificial intelligence planning systems. *Journal of Applied Artificial Intelligence*, 3:1–32, 1989.

[38] N. M. Sadeh. *Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh PA, March 1991.

[39] D.E. Smith, J. Frank, and A.K. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[40] S. F. Smith and C. Cheng. Slack-based Heuristics for Constraint Satisfactions Scheduling. In *Proceedings of the* 11^{th} *National Conference on Artificial Intelligence, AAAI-93*, pages 139–144. AAAI Press, 1993.

[41] S.F. Smith. OPIS: A Methodology and Architecture for Reactive Scheduling. In M. Zweben and S. M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[42] E.P.K. Tsang. *Foundation of Constraint Satisfaction*. Academic Press, London and San Diego, CA, 1993.

[43] S. Wolfman and D. Weld. Combining Linear Programming and Satisfiability Solving for Resource Planning. *Knowledge Engineering Review*, 16(1):85–99, 2000.

Appendix - The DDL.3 Domain Definition Language

The OMPS domain theory is described using DDL.3 specifications. The DDL.3 domain description language is an evolution with respect to our previous proposals called DDL.1 [10] and DDL.2 (described in [9]).

The basic entity of the modeling language is the *Domain*, which is composed of *component type* and *component* definitions[3]:

<PlanningDomain> ::= "DOMAIN" <DomainName>
"{"
(<ComponentTypeDefinition>)*
(<ComponentDefinition>)*
"}"

The *component type* definition requires the specification of (1) the class that actually implements the component, (2) the type name, (3) the values that can be taken by component's behaviors and (4) the consistency function of every component belonging to this type. The actual syntax of such specifications depends on the component:

<ComponentTypeDefinition> ::= "COMP_TYPE" <TypeClass> <TypeName>
<BehaviorDescription>
<ConsistencyFunctionSpecification> ";"

For instance a state variable component definition specifies the behaviors through the list of symbolic values that the state variable may take. Each value is specified with its name and a list of parameter types. Indeed the possible state variable values for DDL.3 are a discrete list of predicate instances like $\mathcal{P}(x_1, \ldots, x_m)$. For each state variable $\mathcal{SV}_i$ we specify (1) a domain $\mathcal{DV}_i$ of predicates $\mathcal{P}(x_1, \ldots, x_m)$ and (2) a domain $\mathcal{DX}_j$ for each parameter x_j in the predicate. Consistency functions for state variables specify (1) the duration of values in every consistent timeline (as a lower and an upper bound) and (2) the transitions allowed among values. The following piece of DDL.3 specification describes a state variable type, GROUND_STATION_VIS_TYPE. Each behavior of a component belonging to this type will be a sequence of alternates values $Visible(?st_i)$ and $None()$ ($\cdots \rightarrow Visible(?st_i) \rightarrow None() \rightarrow Visible(?st_j) \rightarrow \ldots$), where $?st_i$ is a parameter belonging to a type that enumerates the allowed codes for available ground stations. In fact, transition constraints $Visible(?st_i)$ MEETS $None()$ and $None()$ MEETS $Visible(?st_i)$ are specified. No upper bounds are specified for value durations (duration is $[1, +INF]$).

```
COMP_TYPE StateVariable Ground_Station_Vis_Type
  (Visible(GROUND_ST) , None()) {

  VALUE Visible(?st1) [1,+INF]
  MEETS { None() }
```

[3]For the sake of clarity, some other less important features of the modeling language (e.g., *parameter types* and *functions*) will not be described in detail here.

```
    VALUE None() [1,INF]
    MEETS { Visible(?st) }
  };
```

Reusable resource definitions are simpler, since it is not necessary either to enumerate symbolic values which their profiles can take nor to specify transition constraints. In fact it is sufficient to specify the maximum amount max of resource available. A value $A(?q)$ is added by default to the set of values that a decision for this component can take ("A" stands for "Activity", $?q$ is an integer parameter ranging from 0 to max specifying the amount of resource booked by the decision). The following line of DDL.3 code defines a binary resource:

```
COMP_TYPE ReusableResource BIN_RES : 1;
```

The *component* definition requires the specification of (1) the component type the component belongs to and (2) the synchronizations eventually required for component's timelines. A synchronization is a conjunction of component's values requirements (see Definition 5 on Section 4). For instance the following piece of DDL.3 code requires for a value $Transmit(?st)$ a value $Locked(?st)$ (on the same ground station $?st$) for the component POINTING_SYSTEM during its temporal occurrence, and an allocation of $?q1$ units of the BATTERY component exactly in the same temporal interval (the actual amount of the allocation is computed through the function $\#SlewingConsumption$, that returns, at run time the bounds of required amount of battery required for this operation as a function of the duration of the operation itself):

```
VALUE Transmit(?st) {
     %Station must be locked
     DURING [0,+INF] [0,+INF] Pointing_System Locked(?st),
     %Required energy
     EQUALS Battery A(?q1),
     ?q1 = #TransmissionConsumption(?_duration)
   }
```

To give a practical example we include here the DDL.3 formulation of two domains described in this paper: the Rochester Door domain (introduced in Section 3.2) and the Satellite Transmission Domain (introduced in Section 5.1).

The Rochester Door Domain

```
1.  DOMAIN Rochester_Door {
2.
3.
4.    %%%%%%%%%%%%%%%%%%%%%%%%
5.    %%% BEGIN TYPE DEFS %%%
6.    %%%%%%%%%%%%%%%%%%%%%%%%
7.
8.    COMP_TYPE ReusableResource BIN_RES : 1;
9.
10.   COMP_TYPE StateVariable SV_DOOR (Shut(), Open()) {
11.
12.     VALUE Shut() [1,+INF]
13.     MEETS  {Open()}
14.
15.     VALUE Open() [1,+INF]
16.     MEETS  {Shut()}
17.   };
18.
19.   COMP_TYPE StateVariable SV_BIN (Held_Down(), Held_Up()) {
20.
21.     VALUE Held_Down() [1,+INF]
22.     MEETS  {Held_Up()}
23.
24.     VALUE Held_Up() [1,+INF]
25.     MEETS  {Held_Down()}
26.   };
27.
28.
29.   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
30.   %%% BEGIN COMPONENT DEFS %%%
31.   %%%%%%%%%%%%%%%%%%%%%%%%%%%%
32.
33.   COMPONENT Door : SV_DOOR;
34.   COMPONENT Handle : SV_BIN;
35.   COMPONENT Spring_Lock : SV_BIN;
36.   COMPONENT Left_Hand : BIN_RES;
37.   COMPONENT Right_Hand : BIN_RES;
38.
39.   COMPONENT Door: SV_DOOR {
40.     VALUE Open() {
41.       EQUALS Handle Held_Down(),
42.       EQUALS Spring_Lock Held_Up()
43.     }
44.   };
45.
46.   COMPONENT Handle: SV_BIN {
47.
48.     VALUE Held_Down() {
49.       EQUALS Left_Hand A(1)
50.     }
51.
52.     VALUE Held_Down() {
53.       EQUALS Right_Hand A(1)
54.     }
55.   };
56.
57.   COMPONENT Spring_Lock: SV_BIN {
58.
59.     VALUE Held_Up() {
60.       EQUALS Left_Hand A(1)
61.     }
62.
63.     VALUE Held_Up() {
64.       EQUALS Right_Hand A(1)
65.      }
66.    }; }
```

The Satellite Transmission Domain

```
1.  DOMAIN Transmission_System
2.  {
3.    %%%%%%%%%%%%%%%%%%%%%%%%
4.    %%% BEGIN TYPE DEFS %%%
5.    %%%%%%%%%%%%%%%%%%%%%%%%
6.
7.
8.    PAR_TYPE EnumerationParameterType GROUND_ST
9.      {stat1,stat2,stat3,stat4,stat5,stat6};
10.
11.   COMP_TYPE StateVariable Pointing_System_Type
12.     (Slewing(GROUND_ST,GROUND_ST) , Locked (GROUND_ST),
13.       Unlocked (GROUND_ST)) {
14.
15.     VALUE Slewing(?st1,?st2) [1,30]
16.     MEETS { Locked(?st2), ?st1 != ?st2 }
17.
18.     VALUE Locked(?st1) [1,INF]
19.     MEETS { Unlocked(?st1) }
20.
21.     VALUE Unlocked(?st1) [1,INF]
22.     MEETS { Slewing(?st1,?st2) , ?st1 != ?st2}
23.   };
24.
25.   COMP_TYPE StateVariable Ground_Station_Vis_Type
26.     (Visible(GROUND_ST) , None()) {
27.
28.     VALUE Visible(?st1) [1,+INF]
29.     MEETS { None() }
30.
31.     VALUE None() [1,INF]
32.     MEETS { Visible(?st) }
33.   };
34.
35.   COMP_TYPE StateVariable Transmission_System_Type
36.     (Transmit(GROUND_ST) , Idle()) {
37.
38.     VALUE Transmit(?st1) [1,+INF]
39.     MEETS { Idle() }
40.
41.     VALUE Idle() [1,INF]
42.     MEETS { Transmit(?st) }
43.   };
44.
45.   %Max power of the battery: ~900 W,
46.   %level of charge: [486000000,4860000000],
47.   %solar flux reduction factor: 1
48.   COMP_TYPE PowerManagement Available_Power:
49.     [900,[486000000,4860000000],1];
50.   COMP_TYPE SimpleProfile Solar_Panel_Charge: [SFLU_file.mex];
51.
52.   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
53.   %%% BEGIN COMPONENT DEFS %%%
54.   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
55.
56.   COMPONENT Pointing_System : Pointing_System_Type;
57.   COMPONENT Transmission_System:Transmission_System_Type;
58.   COMPONENT Ground_Station_Vis:Ground_Station_Vis_Type;
59.   COMPONENT Battery:Available_Power;
60.   COMPONENT Solar_Flux:Solar_Panel_Charge;
61.
62.   COMPONENT Pointing_System : Pointing_System_Type {
63.
64.     VALUE Locked(?st) {
65.       %Station must be visible
66.       DURING [0,+INF] [0,+INF] Ground_Station_Vis Visible(?st)
```

```
67.     }
68.
69.     VALUE Slewing(?st1,?st2) {
70.       %Required energy for slewing
71.       EQUALS Battery A(?q1),
72.       ?q1 = #SlewingConsumption(?st1,?st2)
73.     }
74.   };
75.
76.   COMPONENT Transmission_System:Transmission_System_Type {
77.
78.     VALUE Transmit(?st) {
79.       %Station must be locked
80.       DURING [0,+INF] [0,+INF] Pointing_System Locked(?st),
81.       %Required energy
82.       EQUALS Battery A(?q1),
83.       ?q1 = #TransmissionConsumption(?_duration)
84.     }
85.   };
86. }
```