A Constraint-Based Approach for Planning and Scheduling Timelines

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
Phone: 39 06 44595 270
Fax: +39 06 44595 243

Abstract

Timeline-based planning represents an effective alternative to classical planning for complex domains which require the use of both temporal reasoning and scheduling features. This paper discusses a constraint-based approach to timeline planning and scheduling by introducing the *component-based* approach of OMPS, an architecture for timeline-driven problem solving which draws inspiration from both control theory and constraint-based reasoning.

The rationale behind the component-based approach derives from classical control theory: the planning and scheduling problem is modeled 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 we present in this paper combines the effectiveness of specific and generic constraint solvers for defining consistent timelines which satisfy the current 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 [59] 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 [56, 55, 15, 43, 32, 65]). The constraint-based approach is shown to be an efficient framework for modeling and solving both temporal and resource allocation problems (see [30, 63, 68, 9, 16]). We therefore propose OMPS, a timeline-based planning and scheduling system which draws inspiration from both control theory and constraint-based reasoning. We present both the theoretical framework behind OMPS, the *component-based* approach where we extend the idea of timeline introducing the concept of *component*, and the architecture of OMPS.

OMPS is an evolution of previous work on a planner called OMP [14, 13, 35], which proposed a uniform framework to integrate Planning & Scheduling. While the OMP experience lead to a proof of concept solver for small scale demonstration, the current development of OMPS has lead to a substantial effort both in re-engineering and in extending our previous work.

This paper is organized as follows: Section 2 presents a brief survey of constrain reasoning, planning and scheduling, placing our work in the current literature. Section 3 presents the definition of *component-based* planning and scheduling, introducing our approach and explaining its key features through a toy example. Sections 4 and 5 present the solving algorithm and the architecture of OMPS, focusing on the constraint-oriented design of the various algorithms and modules. Finally, Section 6 presents a more realistic domain modeling example taken from our experience within the space mission planning context at the European Space Agency.

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 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 [74, 22] for further details).

While CSPs are now widely used and accepted by the scheduling community (see [5] for instance), constraint-based reasoning has had less of an 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., [11]) set of activities to available resources. The formulation of the problem requires the modeling of 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 concerns with the generation of a set of tasks to achieve a given goal (see [39] among others for a more cohomprensive discussion). The tasks to be planned are not know in advance, and this made historically more difficult to exploit classical constraint-based formulations, where variables and constraints in must be specified before solving the problem (see also [7]), steering research toward more sophisticated models like DCSPs (Dynamic Constraint Satisfaction Problems) [54].

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

2.1 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 (often called *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.

Moreover various problems have been addressed through different approaches, like the so called *temporal planning* for instance. The challenge is in choosing decisions by explicitly managing temporal information, i.e., the domain theory does not specify cause-effect relationships as simple precedences among decisions, but also several temporal constraints are required about the durations and separations between decisions. Some existing formalisms have been progressivlely refined to take into account such temporal information, while others were born with embedded temporal features.

However it is possible to distinguish two general 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*. 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 a second approach, named here *timeline-based*, the world is modeled as a collection of entities, each with an associated 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 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 the two following subsections we analyze both approaches in more detail, and provide some insight on the connections of both action and timeline based approaches to CSP techniques.

2.1.1 Action-Oriented Planning

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 an hypothetical executing agent to change the world's state in order to achieve its goals.

The STRIPS formalism [28] was one of the first proposals for action-oriented planning domain modeling. The world state is described as a conjunction of first-order, positive and function-free literals, while actions are specified in the domain theory by stating their *preconditions* (i.e., the states of the world in which they are applicable) and their *effects* (how they modify the truth value of literals in the world state). Actions are assumed to be executed instantaneously and there is no way to deal with any kind of quantitative temporal information: basically it is possible only to reason about what happens before and after an instantaneous action.

The STRIPS formalism has been refined over the years (see ADL [61] and TF [71] for instance), and it is the basis of the most widely used formalism in planning, the PDDL interlingua (see [29, 53]).

Many planners that have been developed are grounded on the same idea of finding a sequence of actions to bring the world from an initial state toward a goal state. STRIPS itself is a planner that performs *means-end analysis* to build a plan (this technique has been used before in GPS [57]). Following the push resulting from the Inter-

national Planning Competition, a considerable number of researchers have addressed the problem of empowering the best planning algorithms to deal with some of the features of scheduling problems. Complete citations are not possible here, but see at least techniques for plan representation and reasoning used in GRAPHPLAN [10], subsequent graphplan-based approaches [46, 64], and "graphplan-like" planners as BLACKBOX [44] or LPG [37], the use of heuristic search [40], the extensions to SAT [76] and to CSP like CPLAN [75] and GP-CSP [24] or several integrations of some of them [23]. These efforts have resulted in a remarkable improvement of performance (due to the fact that attention has focused mainly on algorithmic aspects and search improvements) and an interesting advancement in the expressiveness of the problems addressed (for this last aspect see the PDDL2.1 document [29]).

CSP technologies have been used in action-based planning almost in the same way as SAT-based technolgies have been exploited, i.e., with a sort of "black-box" approach. 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. The basic notion underlying this "black-box" approach is the following[1]: the planner assumes it knows the length of the plan to reach the given goals. It this expands its search space to the point that it can contain such a plan, and encodes this into a CSP representation. The CSP is then solved by a dedicated CSP solver: if no solution is found, the length of the plan is increased, otherwise the CSP solver's solution is decoded back to a set of partially ordered actions, which constitutes a plan for achieving the goal.

The fundamental question in this approach is how to encode the search space into a CSP such that the underlying solver will be able to solve it efficiently. One way of performing this encding is by means of a state variable representation [4]. Each state variable is translated into a CSP variable. To find a plan of length n requires $n + 1$ sets of those variables (the first set representing the initial state and the $i - th$ set representing the state of the world after i actions) plus n CSP variables to track which action is performed at the $i - th$ step. Constraints among those variables are posted to link pre-conditions and effects with actions at each level (a more extensive description of this model can be found in [39]).

A different translation is used by the GP-CSP planner [24], where the search space is encoded into a dynamic constraint satisfaction problem or DCSP ([54]). The encoding of the actions and constraints exploits the underlying structure of the search space (which is represented as a so-called planning graph [10]) by using the idea of suporting actions for given propositions.

A third approach is presented in [52], where the conatraints model a sort of "next state constraint": instead of explicitely modeling the constraints that link an action with its effects, the constraints link directly two layers of the graph, by encoding the relationships between the values of variables of a given level i directly with the values of variables in the level $i-1$ and the value of the action variables (basically the constraints state that a given value is taken in the $i - th$ level either if it is already present at the level $i - 1$ and it is not canceled by an action or if it is asserted by an action).

[1] This methodology was first envisaged in [45]

Despite the differences among them, there is a common problem in the way the CSP is employed: there is not meaning behind the problem encoded, i.e., there is no underlying structure in the encoding, and the CSP solver basically does not know what the variables actually represent at the higher level of planning. It is a sort of "brute-force" approach: the problem is encoded, the CSP solver is triggered, and its solution is eventually translated back to actions.

2.1.2 Timeline-Based Planning

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 th 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 (the timelines).

Being more recent that the action-based approach, timeline-based planning has been less studied and formalized. Indeed, STRIPS-like planning systems exploit well known and studied logical descriptions for world state modeling, action schema descriptions and planning inference. On the other hand, timeline-based planning systems have been built "historically" to address practical problems, and less attention has been devoted clearly formalizing and describing the approach. At present, some timeline-based planning architectures are in use, mainly because they are supported by consistent research groups: RAX-PS [43] and its successors EUROPA and EUROPA2, that inherit the experience of HSTS [56, 55] and ASPEN [19].

This paradigm shares many common elements with control theory: the state variables are basically objects with their own internal behavior, and the problem is to find an input sequence that can be used to obtain desired outputs. The notions of *time* and *concurrency* are embedded into this kind of model of the world. In fact the whole system is split into different objects that concurrently behave in time as a consequence of decisions that are posted over them (see also [59] for a wider disussion about connections between AI planning and control theory).

Though there are many planners that can be clearly classified as action-oriented or timeline-based, there also exist a number of proposals which use intermidiate approaches. The idea of representing the state of the world in the action-oriented approach using state variables [4] as well as modeling directly transitions among the state of the world [52], or a combination of both approaches as described in [41], can be seen as a step of action-oriented planners towards the timeline-based approach.

The enhancement of the classical approach to planning for dealing with time and resources [51, 37, 25, 26, 27, 17] pushed the introduction into action-oriented planning system of tools and results (like temporal networks or scheduling algorithms for instance) which constitute the core of many timeline-based planners. In fact, in a timeline-based perspective the planning problem is naturally temporal (the planning process is performed allocating decisions over timelines) and most of the timeline-based architectures posses scheduling features. Looking from the opposite side, there have been also some attempts to compare the two approaches (see [33, 31, 66] among

the others).

Unlike action-based planners, the timeline-based approach has a more strict connection with CSP techniques. That is because timeline-based approaches explicitly represent temporal information or resource reasoning, features that have a successful CSP-based tradition. Compared to the "black-box" approach, these architectures leverage the CSP model to a larger extent. In some of them 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 planner presented later in this paper, 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.

Among the systems where CSP technology has been exploited to model and solve one or more features of the planning problem it is worth mentioning PARCPLAN [49, 27] (where resource reasoning is implemented as a CSP), IXTET [38] (this is the architecture that more closely follows a CSP approach as a general frame of reference for temporal and resource reasoning), HSTS [56, 55] (designed with a constraint-based scheduling approach). Finally, it is woth mentioning some works which extend this idea, e.g., [32, 34] and OMP [14] (where the CSP approach permeates almost the entire design and solving process).

2.2 Scheduling

Scheduling is primarily concerned with figuring out *when* a set of predefined tasks should be executed so that the final solution guarantees "good" performance relatively to the optimization of an objective function. Let us focus on a family of problems known as *project scheduling*, whose main elements are the following: (a) a set A = $\{a_1, ..., a_n\}$ of *activities* or tasks. Every activity is characterized by a processing time p_i; (b) a set R = $\{r_1, ..., r_m\}$ describing the *resources* required to execute the activities. The execution of each activity ai can require an amount req_{ik} of resource r_k during its processing time. Various kinds of resources can be taken into account: disjunctive or cumulative, renewable or consumable, among others; (c) *constraints*: the constraints are rules that limit the possible allocations of the activities. They can be divided into two types: (1) *resource constraints* limit the maximum capacity of each resource. For example, there may only be a certain number of machines or people available to work on some activities at any given time; (2) *temporal constraints* impose limitations on the times in which activities can be scheduled. A binary constraint is imposed between two activities, for instance in order to mutually bind the instant of occurrence of their start times. Such constraints are often formulated as bounds on differences between activity start/end time points [21]. A scheduling problem consists in computing a consistent assignment of start and end times for each activity satisfying both resource and temporal constraints while optimizing a certain evaluation function, e.g., the early finish time of the whole set of activities or *makespan*. The particular constraints expressed in project scheduling problems are interesting because they allow to model a quite broad range of real domains that require modeling causal relations between activities, coordination between multiple steps, and a rich variety of time and resource constraints.

Scheduling problems are very hard problems: for instance, simple scheduling problems like job-shop scheduling are NP-hard [36]. Therefore, scheduling represents an important application for constraint directed search. The CSP framework involves the combination of sophisticated search technology and constraint propagation. Constraint propagation actively uses constraints to prune the search space. These propagation techniques are generally polynomial w.r.t. the size of the problem, and aim at reducing the domains of variables involved in the constraints by removing the values that cannot be part of any feasible solution. Various techniques with distinct pruning power can be defined for different types of constraints. The search for a solution to a CSP can be viewed as modifying a constraint graph by adding and removing constraints, where the constraint graph is an evolving representation of the search state, and a solution is a state with a single value remaining in the domain of each variable, and all constraints are satisfied. Several constraint programming approaches have been developed in this direction. For instance, the reader can refer to [5] for a thorough analysis of constraint-based techniques for scheduling problems. The work of constraint directed scheduling of the 80s (see for example [63, 68]) developed into Constraint-based Scheduling approaches in the late 90s (see for example [58, 8]). These approaches are based on the representation of a scheduling problem and the search for a solution to it by focusing upon the constraints in the problem. A typical formulation of the problem is that of finding a consistent assignment of start times for each goal activity. Under this model, decision variables are time points that designate the start times of various activities and CSP search focuses on determining a consistent assignment of start time values.

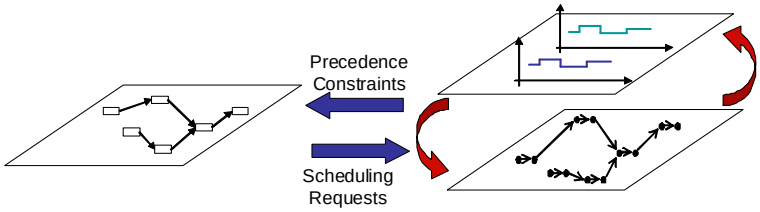

Figure 1: The Precedence Constraint Posting (PCP) approach.

Other approaches to scheduling operate with a problem formulation more akin to least-commitment frameworks. According to this formulation, referred to as *Precedence Constraint Posting* [67, 18], the goal is to post sufficient additional precedence constraints between pairs of activities for the purpose of pruning all inconsistent allocations of resources to activities. A basic method to generate solutions to project scheduling is proposed in [18] for problemswith binary resources, and then extended to more general problems in later work, e.g., [16]. It is is based on the fact that the relevant events on a scheduling problem can be represented as a temporal CSP, usually called Simple Temporal Problem [21]. The search schema used in this approach focuses on decision variables which represent conflicts in the use of the available resources; the solving process proceeds by ordering pairs of activities until all the current

resource violations are removed. This approach is usually referred to as the Precedence Constraint Posting (PCP) because it revolves around imposing precedence constraints to solve the resource conflicts, rather than fixing rigid values to the start times. The general schema of these approaches is described in Figure 1.

It consists in representing, analyzing, and solving various aspects of the problem in two separate layers, namely the temporal layer and the resource layer. In the former, the temporal aspects of the scheduling problem like activity durations, constraints between pairs of activities, due dates, release time, etc., are considered. The latter represents and reasons upon the resource aspects of the problem.

Among the scheduling systems based on CSP techniques it is worth mentioning OPIS [69], SONIA [48], ILOG scheduler [42] and O-OSCAR [12]. The last one in particular deeply influenced the design of OMPS, the planning and scheduling architecture presented here. In fact, O-OSCAR can be easily seen as a sort of timeline-based scheduler: resources are modeled as a collection of entities that evolve in time as a consequence of activity allocations.

2.3 Planning and Scheduling Integration

Current planning and scheduling literature exposes a trend to integrate both planning and scheduling features with the aim of addressing more challenging real-world problems (see for instance [73] and [66]). Since these integration attemps often relied on CSP techniques at some levels, it is worth briefly recalling them here.

Attempts to integrare planning and scheduling have originated both from a planning-centric view, trying to improve planning systems in order to deal with temporal and resource management, and from the scheduling community, in an attempt to improve scheduling systems in order to deal with cause-effect relationships among activities.

From the planning perspective, specialized scheduling capabilities are required in a planning engine as soon as non trivial resource constraints have to be taken into account: even if some approaches have pursued the idea of embedding resource models directly into a planning engine, it is still not clear how complex problems over multi capacity and consumable resources can be afforded without exploiting the capabilities of powerful scheduling reasoners.

On the other hand specialized planning features are required in a scheduling environment in order to deal with complex situations where paying attention only to resource capacity constraints is not enough to solve the problem: it could be useful to perform goal-oriented problem solving or it could be necessary to guarantee some logical properties between activities that cannot be decided in advance when the scheduling problem is formulated. For instance, a resource could require set-up activities whose actual number and nature cannot be decided at the onset of reasoning, but depends on the order that the scheduler chooses for other activities during the problem solving.

Several architectural approaches to integrate temporal planning and resource management in an action-oriented context exist. REALPLAN [70] proposes two levels of integration: a "Master-Slave" approach where the planner controls the scheduler, and a "Peer-to-Peer" approach where both engines work separately and share information using a predefined protocol. A similar architecture is also proposed in [62], while [60]

analyzes the computational aspects of coupling action-oriented planning algorithms with a separate CSP-based scheduler for resource management. In O-PLAN [20] and O-PLAN2 [72], non linear planning techniques are exploited, managing simple temporal and resources constraints as done for action preconditions and effects protection. *parc*PLAN [49] has been improved to handle a STRIPS-like formalism with *temporal propositions* (see [27] and [50]), performing an interval-based planning where CSP problem solving techniques have been used to reason about temporal and resource constraints.

These approaches seem still quite far from obtaining a real integration between planning and scheduling: they certainly improve the expressiveness of planning domain modeling languages to handle quantitative temporal constraints and resource availability management, but the main problem resides in the action-oriented nature of these approaches. It is very difficult in fact to make a planning engine and a scheduling engine work together over a model of the problem that has been originally thought only for one of them. In the best case non trivial adaptations are needed with consequent performance degradation, very often leaving to the problem solver the only possibility of serializing the two tasks. A complete plan produced by the planner is translated into a problem that a scheduler can manage, but in the case of failure there is no direct feedback to the planning engine that could help in fixing the produced plan. At a glance these approaches rather than performing planning and scheduling, perform "planning with temporal and resource reasoning". In other words they have been built to address the need to manage time and resource information, and have not drawn ideas and techniques from the scheduling research area. In fact it is not clear how classical scheduling problems can be solved within these architectures, and the claim that planning *and* scheduling is performed should mean that scheduling problems can be solved in addition to planning problems (or at least modeled).

In the timeline-based context almost all architectures that have been proposed perform temporal and resource management reasoning. They were conceived as temporal planners, thus quantitative temporal constraints management is an embedded feature in these systems, and the timeline-based approach itself facilitates resource reasoning.

Among the planning systems that integrated scheduling features it is worth recalling the already cited RAX-PS and IXTET systems (where the planner drives the computation, exploiting a conflict analyzer to perform resource reasoning) and JPL's ASPEN architecture (where "activities" are the central data structure, representing actions either in a plan or in a schedule).

From the scheduling perspective, other approaches have pursued the idea of extending a scheduling engine with planning capabilities to deal with complex resource models. For instance, Visopt ShopFloor [6] is grounded on the idea of working with dynamic scheduling problems where it is not possible to describe in advance activity sets that have to be scheduled. Dynamic aspects of the problem are modeled using resources with complex behaviors.

Overall, different constraint-based formulations have proved their value in a number of problem solving contexts where either planning, scheduling or both are required. From the computational point of view, constraint-based formulations allow to harness the power of general-purpose CSP solving algorithms (e.g., SAT solvers for graphplan-based planning) as well as more specific constraint-based algorithms such as Simple

Temporal Problem (STP) [21] solvers and PCP algorithms for resource scheduling. This versatility of constraint-based approaches suggests that constraint-based representations and solving algorithms can be effectively used to theorize and build integrated planning and scheduling solving tools. This is, indeed, the objective of OMPS.

3 The Component-Based Approach

As menioned, 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. Decisions can be taken on components to alter their behaviors, and causal constraints between decisions describe how the evolutions of the system as a whole can be altered. Components provide the means to achieve modular decision support tool development. A component can be designed to incorporate entire decisional modules which have been developed independently into the planning and scheduling framework.

In addition to being grounded on control theory, the OMPS approach is strongly CSP-oriented: temporal reasoning, plan construction and scheduling procedures are modeled by means of variables and constraints among them. The general modeling philosophy is constriant-based as well: the model is built by identifying relevant features of the domain and expressing constraints about their concurrent temporal evolutions.

3.1 Components and Behaviors

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

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

It is in general possible to have many different representations of a component's consistency features: either explicit (e.g., tables or allowed bounds) or implicit (e.g., constraints, assertions, and so on). For instance, let us model a light bulb component.

A light bulb's behaviors can take three values: "on", "off" and "burned". Supposing the light bulb cannot be fixed, a rule could state that any behavior that takes the value "burned" at a time t is consistent if and only if such a value is taken also for any time $t' > t$. This is a declarative approach to describing the consistency feature f^C. Different actual representations for this function can be used, depending also on the representation of the behavior.

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

State variable. *Behaviors*: piecewise constant functions over a finite, discrete set of symbols which represent the *values* that can be taken by the state variable. Each behavior represents a different sequence of values taken by the component. *Consistency Features*: a set of sequence constraints, i.e., a set of rules that specify which transitions between allowed values are legal, and a set of lower and upper bounds on the duration of each allowed value. The model can be for instance represented as a timed automaton [3] (e.g., the three state variables in Figure 6). Note that a distinguishing feature of a state variable is that not all the transitions between its values are allowed.

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

The component-based nature of OMPS allows to accommodate complex or pre-existing solving components into larger planning contexts. In the example described in Section 6 for instance, a *battery component* will be presented. This component has been built by extending the modeling power of a renewable resource component.

3.2 Component Decisions and Domain Theory

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

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

Regardless of the type of component, the value of any component decision can contain *parameters*. In OMPS, parameters can be numeric or enumerations, and can be used to express complex values, such as "transmit(?bitrate)" for a state variable which models a communications system.

Overall, a component decision is something that happens somewhere in time and modifies a component's behaviors as described by the value ν. In OMPS, the conse-

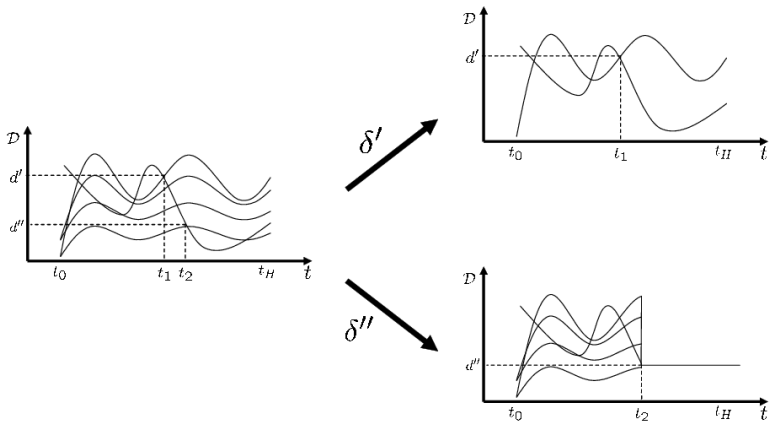

Figure 2: The update function computes the results of a decision on a component's set of behaviors. The figure exemplifies this effect given the two decisions: δ' imposes a value d' for the behaviors of the component in the time instant t_1; δ'' imposes that the values of all behaviors converge to d'' after time instant t_2.

quences of these decisions are computed by the components by means an *update function* f^U. This is a function which determines how the component's behaviors change as a consequence of a given decision. In other words, a decision changes a component's set of behaviors, and f^U describes how. A decision could state for instance "keep all the behaviors that are equal to d' in t_1" and another decision could state "all the behaviors must be equal to d'' after t_2". Given a decision on a component with a given set of behaviors, the update function computes the resulting set (see Figure 2).

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

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

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

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

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

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

In order to clarify how such inter-component relationships are modeled as a domain theory, let us give an example, showing a formalization of a well-known domain introduced by Allen in [2].

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 open with one hand, while the door is pulled open 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 up. 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 is quite challenging. For the problems involving classical planners and an alternative solution with respect to the present one, the interested reader is refered 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, thus we model the domain using a state variable type SV_BIN and a reusable resource type BIN_RES with capacity 1. State variables belonging to the type SV_BIN will take values *Open()* or *Shut()*, switching from one to the other.

In OMPS the goal of opening the door is seen as a constraint imposing 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 *Open()* (in therms of synchronizations $\langle$DOOR,*Open()*$\rangle \rightarrow \langle$HANDLE,*Open()*,$\{$DURING$\}\rangle$ and $\langle$DOOR,*Open()*$\rangle \rightarrow \langle$SPRING_LOCK,*Open()*,$\{$DURING$\}\rangle$). Moreover both actions to open the handle and the spring lock require using hands: this can be stated in OMPS terms by specifying the additional constraint that when the state variables HANDLE and SPRING_LOCK take the value *Open()*, they also require using either resource LEFT_HAND or resource RIGHT_HAND. Appendix 7 shows the formulation of this domain for OMPS.

4 Timeline-Driven Problem Solving

In order to reason about components and their behaviors, we need to provide a means to represent and inspect the outcome of component decisions on the components' behaviors. To this end, we formulate the notion of *timeline*. A timeline is defined as an ordered sequence of a component's 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.

The overall solving process implemented in OMPS is composed of three main steps, namely *domain theory application*, *timeline management* and *solution extraction*. Indeed, a fundamental principle of the OMPS approach is its *timeline-driven* solving process. The inputs of this process are (1) a component based domain with a domain theory, (2) a set of *initial conditions*, and (3) a set of *goals*. The output of the solving process is a set of timelines, one for each component in the domain.

Initial conditions are decisions that do not lead to application of the domain theory. In other words, initial conditions describe pieces 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 in a set of component decisions which are intended to trigger the solving strategy to exploit the domain theory's synchronizations to synthesize decisions. This mechanism is the core of the solving process described in the following sections.

4.1 Domain Theory Application

The fundamental data structure in OMPS's timeline-driven solving process is the *decision network*: given a set of components $\mathcal{C}$, a decision network is a graph $\langle V, E \rangle$, where each vertex $\delta_C \in V$ is a component decisions defined on a component $C \in \mathcal{C}$, 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 $\delta^n_{C^j}$.

Dependencies among component decisions are specified by means of *relations*, the edges of the decision network, of which OMPS provides three types; *temporal*, *value* and *parameter* relations.

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

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

Lastly, a *parameter relation* among component decisions is a constraint among the values of the parameters of the two decisions. Such relations can prescribe linear inequalities between parameter variables. For instance, a parameter constraint between

two decisions with values "available(?antenna, ?bandwidth)" and "transmit(?bitrate)" can be used to express the requirement that transmission should not use more than half the available bandwidth, i.e., ?bitrate $\leq 0.5 \cdot$?bandwidth.

Component decisions possess an attribute which changes during the solving process, namely whether or not a decision is *justified*. OMPS's domain theory application step consists in iteratively tagging decisions as justified according to the following rules (iterated over all decisions δ in the decision network):

1. If δ *unifies*[2] *with another decision in the network.*, then δ can be marked as justified;
2. If δ's value *unifies*[3] *with the reference value of a synchronization in the domain theory*, then δ can be marked as justified provided that the target decision(s) and relations are added to the decision network (as not justified);
3. If δ *does not unify with any reference value in the domain theory*, mark δ as justified.

4.2 Timeline Management

Timeline management is a collection of procedures which are necessary to go from a decision network to a completely instantiated set of timelines. These timelines ultimately represent a solution to the planning problem.

The reason for which a timeline management process is needed lies in the flexible temporal allocation of decisions. In fact, the outcome of the domain theory application step is a decision network where all decisions are justified but, since 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) [21], these decisions are not fixed in time, rather they are free to move between the temporal bounds obtained as a consequence of the temporal relations imposed on the temporal elements. An STP is a constraint-based representation where variables represent time points 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. Constraints between time points are binary, and specify the allowed distance between the involved time points. Formally, given two time points t_i^p and t_j^p, a constraint between them in an STP is an interval $[min, max]$ such that $min \leq t_j^p - t_i^p \leq max$. 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 time points distance. The solution obtained by selecting for each time point its lower bound value is called *earliest start time solution* (EST solution shortly).

[2] 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.

[3] A decision unifies with the reference value of a synchronization if a value constraint EQUALS can be posted between the value of the decision and the reference value of the synchronization.

Timeline management may introduce new component decisions as well as new relations in the decision network. For this reason, the OMPS solving process iterates domain theory application and timeline management steps until the decision network is fully justified and a consistent set of behaviors can be extracted from all component timelines. The specific procedures which compose timeline management are *timeline extraction*, *timeline scheduling* and *timeline completion*. Let us describe these three steps in detail.

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.

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

Figure 3 shows the timeline extraction mechanism for a state variable. The example illustrates two properties of timelines, namely *flaws* and *inconsistencies*. 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 figure). A flaw is a segment of time in which no decision has been taken, thus the state variable within this segment of time is not constrained to take on certain values, rather it can, in principle, assume any one of its allowed values. The process of deciding which value(s) are admissible with respect to the state variable's internal consistency features (i.e., the component's f^C function) is clearly a non-trivial process. Indeed, this is precisely the objective of *timeline completion*.

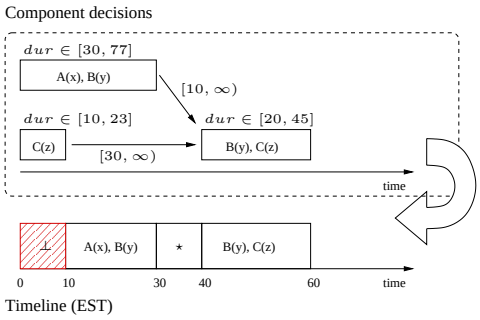

Figure 3: Three value choices on a state variable, and the resulting earliest start time (EST) timeline.

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 cannot present flaws (an interval of time in which no decision is taken implies that the resource is simply unused).When invoked, timeline completion adds new decisions to the plan.

In addition to flaws, inconsistencies can arise in the timeline (as in the interval $[0, 10]$ in Figure 3). 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 value choices whose intersection is empty overlap in time. As opposed to flaws,

inconsistencies do not require the generation of additional component decisions, rather they can be resolved by posting further temporal constraints. For instance, two decisions (A(x), B(y)) and (C(z)) which overlap in time lead to an inconsistency which can be resolved by imposing a temporal relation which forces (C(z)) to occur either after or before (A(x), B(y)). In the case of a resource component, an inconsistency occurs when allocated activities requires more than available resource in a temporal interval. In general, we call the process of resolving inconsistencies *timeline scheduling*.

The scheduling process deals with the problem of resolving inconsistencies. Once again, 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 of decision that requires a conflicting set of decisions must be avoided. The timeline scheduling process adds constraints to the decision network to avoid such inconsistencies through a precedence constraint posting algorithm (described in Section 2.2).

4.3 Solution Extraction

Once domain application and timeline management have successfully converged on a set of timelines with no inconsistencies or flaws, the next step is to extract from the timelines one or more consistent behaviors. A behavior is one particular choice of values for each temporal segment in a component's timeline. The previous domain theory application and timeline management steps have filtered out all behaviors that are not, respectively, consistent with respect to the domain theory and the components' consistency features. Solution extraction deals with the problem of determining a consistent set of fully instantiated behaviors for every component. Since every segment of a timeline potentially represents a disjunction of values, solution extraction must choose specific behaviors consistently. Furthermore, not all values in timeline segments are fully instantiated with respect to parameters, thus solution extraction must also take into account the consistent instantiation of values across all components.

4.4 Overall Solving Process

In the current OMPS solver the previously illustrated steps are interleaved as sketched in Figure 4. The first step in the planning process is domain theory application, whose aim is to support non-justified decisions. If there is no way to support all the decisions in the plan, the algorithm fails.

Once every decision has been supported, the solver tries to extract a timeline for each component. 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. If the scheduler cannot solve all conflicts, the solver backtracks directly to domain theory application, and searches for a different way of supporting goals.

If the solver manages to extract a conflict-free set of timelines, it then triggers a timeline-completion step on any timeline which is found to have flaws. It may happen that some timelines cannot be completed. In this case, the solver backtracks again to the previous domain theory application step, and again searches for a way of justifying

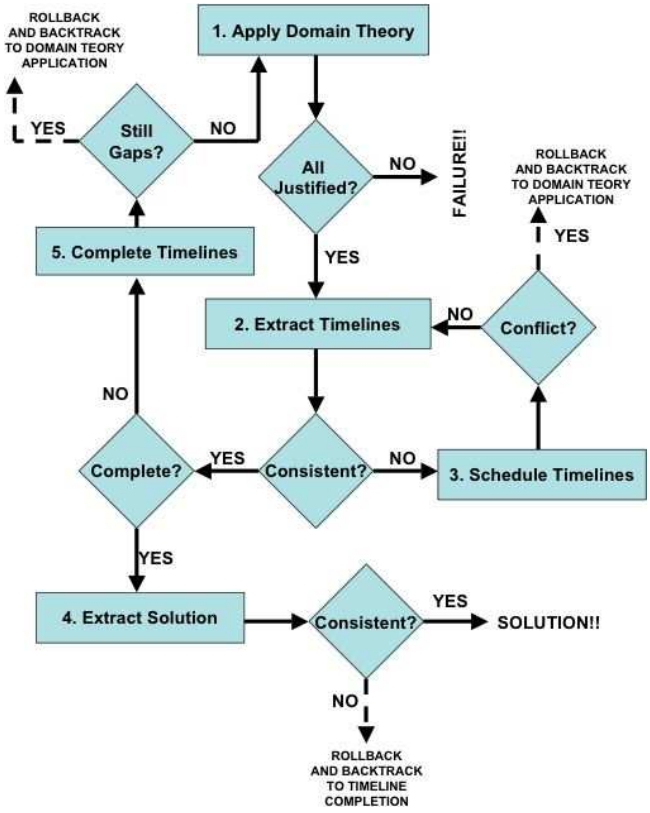

Figure 4: The OMPS solving process.

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 conflict-free and complete, the solver is ready to extract behaviors. If behavior extraction fails, the solver attempts to backtrack to timeline completion. 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 — where a decision can be justified either because it leads to a synchronization with another component or because it unifies with another decision which is already present; the solving process then turns to each component's timeline to see the results of the previous step, and isolates inconsistencies and flaws — this is the core of the solving process, as

timelines provide a "time-instantiated view" of the current decision network; lastly, if timeline management has lead to the introduction of new decisions (which are not justified), the iteration starts over.

Once this iterative process has reached a stable point – where all decisions are justified and there exists a timeline for each component that has no flaws or inconsistencies – the 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 paramteres. If this step terminates succesfully, the set of all obtained behaviors represents a solution, otherwise the planner must backtrack to domain theory application.

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 interweaved with the more "planning-oriented" steps (domain theory applciation and timeline completion), thus making OMPS an example of strongly integrated planning and scheduling.

5 The OMPS Software Architecture

In the previous sections we have introduced the concept of timeline and shown an iterative process for manipulating them in order to solve a given planning and scheduling problem. We now briefly sketch our implementation of the OMPS planning and scheduling framework. As the general approach is centered on timeline management and organization of the domain into components, so is its implementation.

The OMPS software architecture consists of *reasoners* organized in a hierarchy. Each reasoner is responsible for dealing with a particular aspect of the planning and scheduling problem. The constraint-based nature of the approach is extremely visible in the way the different levels exchange information. Specifically, each OMPS reasoner has three capabilities: first, it calculates the effects of *constraints* on *elements* belonging to its "level" in the reasoner hierarchy (for instance a resource can compute the profile resulting from activities allocated on it); second, reasoners can also impose constraints on elements belonging to the lower levels of the hierarchy or create elements on reasoners at lower levels (for instance a resource uses the temporal module to keep track of the temporal occurrence of activities, posting temporal constraints as a result of the scheduling processes). Third reasoners can forward back constraints to higher levels of the hierarchy, as a result of computation performed (for instance, if a higher level reasoner has allocated activities on a resource, the resource can return to this higher level reasoner some precedence constraints that must occur among the activities to avoid over- or under-consumption of the resource).

OMPS has four types of reasoners: a *TemporalModule*, a *Component*, a *DomainManager*, and a *Solver*. Reasoners are organized according to the hierarchy shown in Figure 5.

The **TemporalModule** computes the effects of *temporal assertions* over a set of *temporal elements*. It is at the bottom of the hierarchy and does not impose any assertion

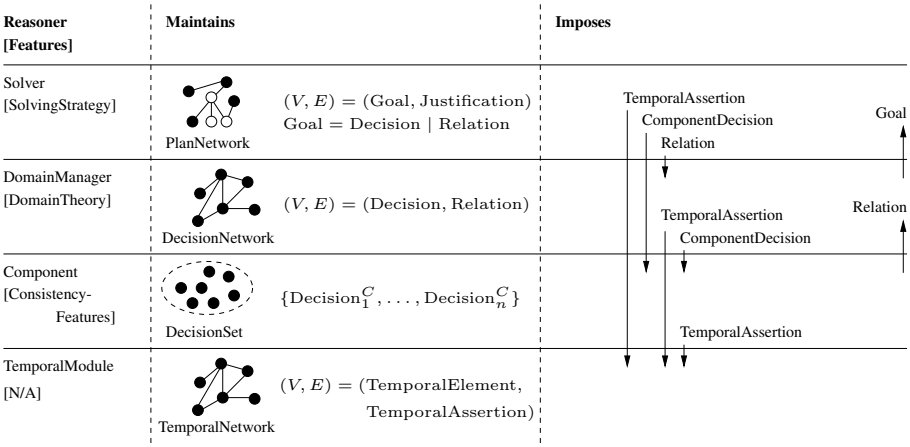

Figure 5: Hierarchy of Reasoners in APSI.

on reasoners at a lower level, nor does it return assertions back to the level above it. The temporal module maintains a data structure, the temporal network, that is a graph: the nodes are *t*ime points, the edges are constraints among them. In the current implementation of OMPS, the chosen temporal model is the STP.

The **Component** is a software module that computes the effects of *decisions* generated by higher levels over its *behaviors*. A component has some consistency features that it uses to distinguish which behaviors are consistent and which are not. On the basis of those features it can forward back to its higher levels *relations* among decisions that have been used to update its behaviors. Also, it can impose temporal constraints over temporal elements.

Let us think of a resource for instance. Decisions over a resource are activities allocated to book a certain amount of available resource. A resource, computing the effects of a set of allocations, generates necessary temporal assertions among start and end times of its activities to prevent an overbooking (the consistency feature in this case says that each allocation profile must be under a given level). These assertions are made by the component on the underlying temporal module. A resource may also be smart enough to have embedded scheduling capabilities. In this case it could compute a set of precedence relations between its activities that, when posted, could ensure that no overbooking occurs. Those relations are forwarded back to the higher module that allocated the activities in the first place, thus allowing it to make informed decisions as to which activities to drop in case of over-consumption.

A **DomainManager** computes the effects of *relations* over a set of *decisions* maintaining the *decision network*. A domain manager can impose or retract both decisions on components as well as temporal assertions on temporal elements. A domain manager has a domain theory it can use to distinguish which of its evolutions are consistent and which are not. On the basis of this theory it can subgoal and forward back to higher

levels goals that have to be achieved.

Let us think for example of a domain with one resource and one state variable. The domain theory says that when the state variable takes a value $P(x)$ an activity $A(q)$ to book x units of resource must be allocated over the same temporal interval. Thus there is both a temporal relationship between the decision $P(x)$ and $A(q)$ as well as a relation between its parameters ($q = x$ in this case). Let us suppose the decision network is updated by creating a new node, the decision $P(x)$. The domain manager posts this decision over the state variable on the level below, creating necessary temporal elements and posting necessary temporal assertions over them in the underlying temporal module. Posting these decisions may result in some relations back from the component level below. Having a domain theory, the domain manager sends back to the higher levels subgoals ($A(q)$, $x = q$ and the temporal relationship between $A(q)$ and $P(x)$). When the plan is updated with a new relation the domain manager translates this relation into temporal assertions and/or new decisions: having for instance $P(x)$ and $A(3)$ in the plan, the relation $x = y$ results in turning $P(x)$ into $P(3)$.

A **Solver** keeps track of *justifications*, i.e., groups of goals, which in turn are either decisions or relations among decisions. A solver maintains a data structure, the *plan network*, that is a hypergraph where nodes are goals and edges are justifications among them. A solver can impose and retract relations on sets of decisions, decisions on components, and temporal assertions on temporal elements. A solver has a solving strategy that drives it in building the plan network (i.e., how to impose relations, decisions, etc. on the lower levels).

Let us clarify with some examples. Let us think for instance of a planner for state variables with a very simple solving strategy: at each step the two choices are available: a domain theory expansion or a unification with an already present decision. The hypergraph thus collapses into a search three since the solver can do only two things, namely decide to unify a decision or to expand the domain theory. In other words, the solver can derive a relation between decisions which are already present in the plan network, or it can decide to expand the domain theory, thus potentially determining the presence of further decisions in the plan network. Specifically, the former behavior consists in the imposition of relations which prune the search space from inconsistencies, while the latter corresponds to the imposition of a "cause-effect" relationship described in the domain theory.

In the case of an integrated planner and scheduler there is one more option: to plan or to schedule, i.e., it is possible to expand the domain theory, to unify to avoid new decisions or to schedule already present decisions. In this case a meta relation could link all the relations added during a scheduling step.

A last example could be a planner that works with resources that can be replenished. The key point is the following: it may be necessary to produce "out of the blue" a given amount of a resource. In this case the planner may directly generate decisions to support some consumptions which are not entailed by domain theory expansions. In this case an edge could link the productions introduced with the consumptions that pushed the decision.

6 Example

OMPS is currently used as a core module of the Advanced Planning and Scheduling Initiative (APSI), a project funded by the European Space Agency (ESA) whose goal is to provide a development environment for designing and implementing space mission planning decision support systems. The aim of using OMPS within APSI is to generalize the approach to mission planning decision support by creating a software framework that facilitates product development. We present here an example inspired from current practice ESA's 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 Appendix 7. Specifically, we define the following five components:

PS, TS and GSV. The spacecraft's pointing and transmission systems, as well as station visibility are modeled with three state variables. The consistency features of these state variables (possible states, bounds on their duration, and allowed transitions) are depicted in Figure 6. They are expressed in the OMPS formalism in the definition of the component *types*, lines 11–43. 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 the figure: one states that the value "locked(?st3)" on component PS requires the value "visible(?st6)" on component GSV (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 (lines 79–80); lastly, both slewing and transmission entail the use of a constant amount of power from the battery (lines 69–74 and 81–83).

SF. The solar flux 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 given

function time $\mathrm{sf}(t)$ which is not subject to changes. This function is provided in a file by ESA and it is used to read in the consistency features 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 features are 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 shows the timelines of the GSV and TS components resulting from the application of a set of initial condition (no initial decision is specified for the PS component). Notice that the GSV timeline 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.

The TS timeline contains five "transmit" value choices, through which we represent our goal (not shown in the Figure). 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 flaws, and it is precisely these flaws 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 that 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 behaviors extracted from the TS and PS components' timelines after applying this solving procedure on our example are shown in Figure 8.

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

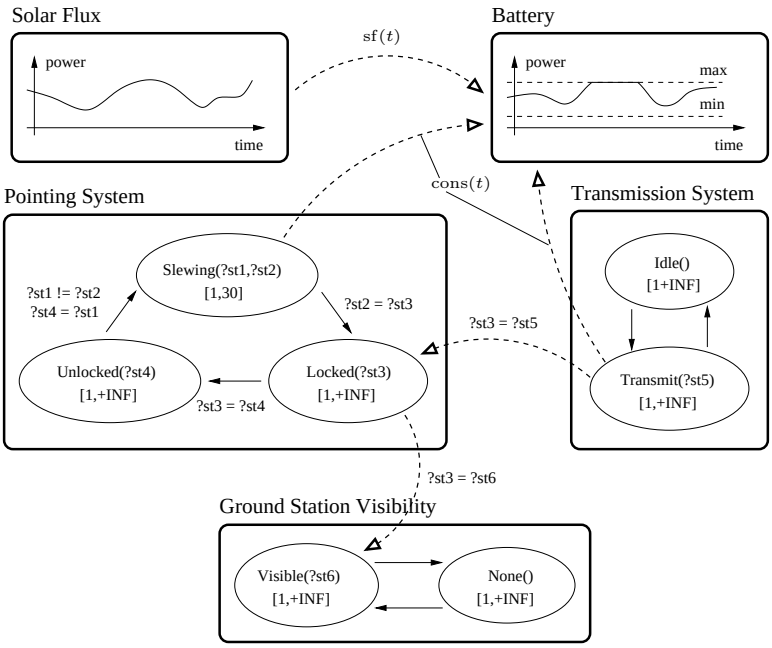

Figure 6: State variables and domain theory for the running example.

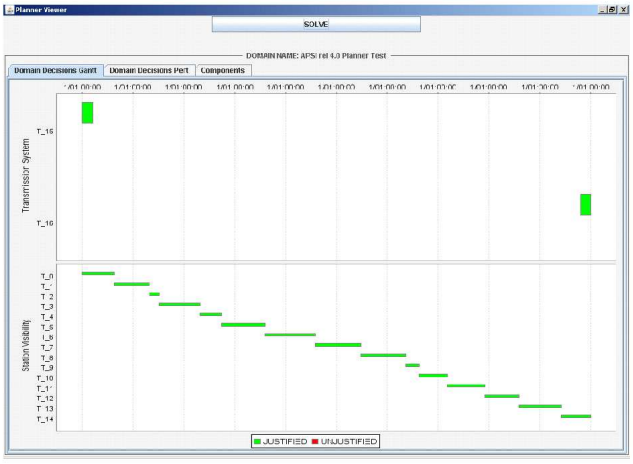

Figure 7: EST timelines for the TS and GSV state variables.

an activity on a reusable resource entails that the resource's availability is decreased by the value of the activity), the BAT's update function calculates the consequences of activities as per the above integration over the planning horizon.

The ability to encapsulate potentially complex modules within OMPS components

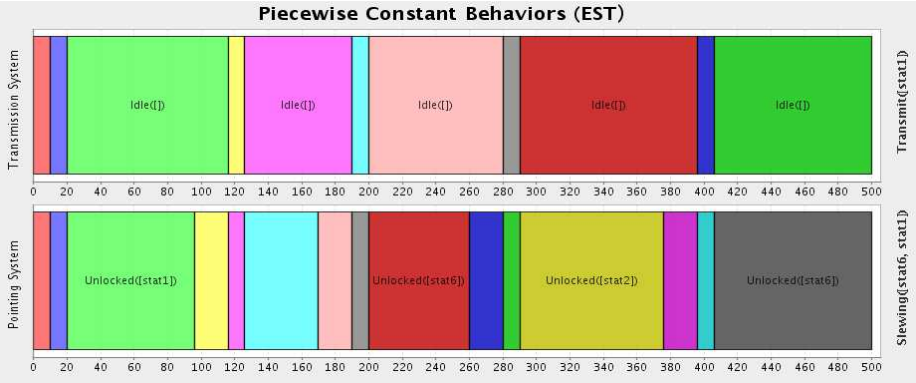

Figure 8: Behaviors for the TS and PS state variables.

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.

7 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 scheduling into a homogeneous solving framework. The result represents, in our view, a structured combination of various proposals present in the literature: on one hand, we employ ideas from other timeline-based planning architectures, such as HSTS [56, 55]; on the other, we integrate advanced constraint-based scheduling features [47, 16] within the very core of the planning process.

The overall approach generalizes the idea proposed in OMP [13] by introducing the concept of components, i.e., any set of properties that vary in time. This includes "classical" concepts such as state variables [56, 55, 15, 43, 32]), as well as renewable/consumable resources [16]. 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 (a particularly important feature in the space domain) 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. The Authors are currently supported by 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.

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] C. Bäckström. *Computational Complexity of Reasoning About Plans*. PhD thesis, Linköping University, 1992.

[5] 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.

[6] R. Bartak. Visopt ShopFloor: On the edge of planning and scheduling. In P. van Hentenryck, editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP2002)*, LNCS 2470, pages 587–602. Springer Verlag, 2002.

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

[8] J.C. Beck, A.J. Davenport, E.D. Davis, and M.S. Fox. The odo project: Toward a unified basis for constraint-directed scheduling. *Journal of Scheduling*, 1:89–125, 1998.

[9] 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.

[10] Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997.

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

[12] A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi. A Constraint-Based Architecture for Flexible Support to Activity Scheduling. In *LNAI 2175*, 2001.

[13] A. Cesta and S. Fratini. Controlling Complex Physical Systems Through Planning and Scheduling Integration. In M. Ali and F. Esposito, editors, *IEA/AIE 2005*, number 3533 in Lecture Notes in Artificial Intelligence, pages 197–207. Springer, 2005.

[14] 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.

[15] 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.

[16] 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.

[17] Yixin Chen, Benjamin W. Wah, and Chih-Wei Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *J. Artif. Intell. Res. (JAIR)*, 26:323–369, 2006.

[18] 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.

[19] 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.

[20] K.W. Currie and A. Tate. O-Plan: Control in the Open Planning Architecture. *Artificial Intelligence*, 51:49–86, 1991.

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

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

[23] M. B Do and S. Kambhampati. *Sapa*: A Domain-Independent Heuristic Metric Temporal Planner. In *Proceedings of ECP-01*, 2001.

[24] 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.

[25] Brian Drabble and Richard Kirby. Associating A.I. Planner Entities with an underlying Time Point Network. In *proceedings of the First European Workshop on Planning*, LNAI 522, pages 27–38. Springer Verlag, 1991.

[26] Brian Drabble and Austin Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner (poster paper). In *Proceedings of the Second International Conference on Planning Systems (AIPS-94)*, 1994.

[27] 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.

[28] 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.

[29] M. Fox and D. Long. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

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

[31] J. Frank, K. Golden, and A. Jonsson. The Loyal Opposition Comments on Plan Domain Description Languages. In *Proceedings of the Workshop on PDDL (ICAPS'03)*, Trento, Italy, June 2003.

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

[33] J. Frank and A. Jonsson. Constraint based attribute and interval planning. *Journal of Constraints*, 8(4):339–364, 2003.

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

[35] S. Fratini. *Integrating Planning and Scheduling in a Component-Based Perspective: from Theory to Practice*. PhD thesis, University of Rome "La Sapienza", Faculty of Engineering, Department of Computer and System Science, 2006.

[36] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, 1979.

[37] 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.

[38] 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.

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

[40] P. Haslum and H. Geffner. Heuristic Planning with Time and Resources. In *Proceedings of ECP-01*, 2001.

[41] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[42] ILOG. <http://www.ilog.com>. ILOG Web Site, 1987.

[43] 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.

[44] 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.

[45] 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.

[46] J. Koehler. Planning under Resource. In *Proceedings of ECAI-98*, 1998.

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

[48] C. Le Pape. Scheduling as Intelligent Control of Decision-Making and Constraint Propagation. In M. Zweben and S. M. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[49] 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.

[50] Vassilis Liatsos. *Scaleability in Planning with Limited Resources*. PhD thesis, University of London, 2001.

[51] Derek Long and Maria Fox. Exploiting a graphplan framework in temporal planning. In *ICAPS*, volume 2, pages 52–61, Seattle, Washington, USA, August 2003. AAAI Press/MIT Press.

[52] Adriana Lopez and Fahiem Bacchus. Generalizing graphplan by formulating planning as a csp. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-2003)*, pages 954–960, 2003.

[53] Drew Mc Dermott, M. Ghallab, A. Howe, C.A. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Tehnical report, CVC TR-98-003 / DCS TR-1165, Yale Center for Communicational Vision and Control, October 1998.

[54] 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.

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

[56] 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.

[57] A. Newell and H. A. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and thought*, pages 279–293. McGraw-Hill, New York, 1963.

[58] W. P. M. Nuijten and E. H. L. Aarts. A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European Journal of Operational Research*, 90(2):269–284, 1996.

[59] 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.

[60] F. Pecora and A. Cesta. Evaluating Plans through Restrictiveness and Resource Strength. In *WIPIS-05. Proceedings of the 2nd Workshop on Integrating Planning into Scheduling at AAAI-05, Pittsburgh, USA*, 2005.

[61] E. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *kr89*, pages 324–332, 1989.

[62] María Dolores Rodriguez-Moreno, Angelo Oddi, Daniel Borrajo, and Amedeo Cesta. IPSS: A hybrid approach to planning and scheduling integration. *IEEE Transactions on Knowledge and Data Engineering*, 18(12):1681–1695, December 2006.

[63] 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.

[64] D. Smith and D. Weld. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of IJCAI-99*, 1999.

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

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

[67] 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.

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

[69] S.F. Smith, P.S. Ow, J. Potvin, N. Muscettola, and D.C. Matthys. An Integrated Framework for Generating Factory Schedules. *Journal of the Operational Research Society*, 41(6):539–552, 1990.

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

[71] A. Tate, B. Drabble, and J. Dalton. *O-Plan Version 2.3 Task Formalism Manual*. AIAI, University of Edinburgh, 1995.

[72] A. Tate, B. Drabble, and R. Kirby. O-Plan2: An Open Architecture for Command, Planning, and Control. In M. Zweben and M. S. Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[73] Austin Tate. The emergence of "standard" planning and scheduling system components – open planning and scheduling architectures. In C. Bäckström and E. Sandewell, editors, *Current Trends in AI Planning: Proceedings of the 2nd European Workshop on Planning (EWSP-93)*, pages 14–32, Vadstena, Sweeden, December 1993. IOS Press (Amsterdam).

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

[75] Peter van Beek and Xinguang Chen. CPlan: A constraint programming approach to planning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-99)*, pages 585–590, 1999.

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

Appendix A - The Rochester Door Domain

```
1.  DOMAIN Rochester_Door {
2.
3.    COMP_TYPE ReusableResource BIN_RES : 1;
4.
5.    COMP_TYPE StateVariable SV_BIN (Shut(), Open()) {
6.
7.      VALUE Shut() [1,+INF]
8.      MEETS  {Open()}
9.
10.     VALUE Open() [1,+INF]
11.     MEETS  {Shut()}
12.   };
13.
14.   COMPONENT Door : SV_BIN;
15.   COMPONENT Handle : SV_BIN;
16.   COMPONENT Spring_Lock : SV_BIN;
17.   COMPONENT Left_Hand : BIN_RES;
18.   COMPONENT Right_Hand : BIN_RES;
19.
20.   COMPONENT Door: SV_BIN {
21.     VALUE Open() {
22.       EQUALS Handle Open(),
23.       EQUALS Spring_Lock Open()
24.     }
25.   };
26.
27.   COMPONENT Handle: SV_BIN {
28.
29.     VALUE Open() {
30.       EQUALS Left_Hand 1
31.     }
32.
33.     VALUE Open() {
34.       EQUALS Right_Hand 1
35.     }
36.   };
37.
38.   COMPONENT Spring_Lock: SV_BIN {
39.
40.     VALUE Open() {
41.       EQUALS Left_Hand 1
42.     }
43.
44.     VALUE Open() {
45.       EQUALS Right_Hand 1
46.      }
47.    };
48. }
```

Appendix B - 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.       EQUALS 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. }
```