The Integration of Planning into Scheduling with OMP

Simone Fratini and Amedeo Cesta

Planning and Scheduling Team [PST]
ISTC-CNR — Institute for Cognitive Science and Technology — Italian National Research Council
Via S. Martino della Battaglia 44, I-00185 Rome, Italy — name.surname@istc.cnr.it

Abstract

This paper describes features of the OMP problem solver that integrates planning features in a constraint based scheduling framework. The rationale for this integration is to express problems going beyond the vision of scheduling as a set of resource productions and consumptions but for also being able to express several kind of causal relations between and behind them. This work describes some steps in the direction of uniformly managing planning as a special case of task scheduling in a constraint-based environment.

Introduction

Current planning and scheduling (P&S) literature exposes a trend to integrate both planning and scheduling features in the aim of addressing more challenging real-world problems.

Even if planning and scheduling have been usually handled independently using different methods and technologies, it could be easy to find strict connections between them. While in certain application domains the subdivision of the two problems as separate entities is quite motivated (see for example (Srivastava, Kambhampati, & Do 2001; Pecora & Cesta 2005)), in other domains such a clear separation of the planning and scheduling phase is more questionable.

From one hand specialized scheduling capabilities are required in a planning engine as soon as non trivial resource constraints have to be taken in account: even if some approaches have pursued the idea to embed resource models directly into a planning engine, it is still not clear how non trivial problems over multi capacity and consumable resources can be afforded without exploiting the powerful capabilities of scheduling reasoners. From the other hand planning capabilities are required in a scheduling environment in order to deal with complex situations where paying attention only to resources capacity constraints is not enough to solve the problem. Very often, it could be necessary to guarantee also some logical orders between the activities that cannot be decided in advance when the scheduling problem is initially formulated. For instance, in several domains a resource could require set-up activities whose position cannot be decided all at once but rather depends on the order that the scheduler chooses for other activities during problem solving.

Several architectural approaches to integrate planning and scheduling problems exists: for instance O-PLAN (Currie & Tate 1991), IxTeT (Laborie & Ghallab 1995), HSTS (Muscettola *et al.* 1992), RAX-PS (Jonsson *et al.* 2000), or ASPEN (Chien *et al.* 2000)) have already succeeded in including aspects from both Planning and Scheduling (P&S) among their features. These architectures have always emphasized the use of a rich representation planning languages to capture complex characteristics of the domain involving time and resource constraints. From the opposite 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 (Bartak 2002) manage resources with complex behaviors described using states and transitions between states. The present work is situated between these two perspectives: mainly we extend a scheduling engine in order to deal with causal relationships between activities that have to be scheduled, but maintain a declarative approach to the problem as typical in AI planning. We are working on OMP, an architecture where a Domain Definition Language (DDL) and a Problem Definition Language (PDL) are used to describe an integrated P&S scenario and a problem that have to be solved in this scenario stated as a set of goals that a planner has to achieve over time. Our aim is to show how the integrated problem can be solved using a scheduling engine able to deal with both classical multiple capacity resources and *state variables* that are described as finite state machines. Roughly speaking an activity for a state variable specify a piecewise constant temporal function on time intervals. On each time interval a state is required to occur for a state variable. To decide correct temporal behaviors for state variables, a planning problem is solved generating a network of activities that matches the pattern that specifies which transition are allowed for state variables and then scheduling these activities. It is necessary to schedule them because the patterns that have to be matched in general could generate partially ordered set of activities, as we will show better in the following, so these networks have to be scheduled in order to avoid that activities requiring contradictory states could overlap.

By means of a constraint based representation, OMP uniformly deals with causal and resource constraints "on top" of a shared layer representing temporal information as a Simple

Temporal Problem (STP) (Dechter, Meiri, & Pearl 1991). For the planning domain description we use a representation of domain components (that we call *state variables* as said before), consisting in temporal automata, as first proposed in HSTS (Muscettola *et al.* 1992; Muscettola 1994) and studied also in subsequent works (Cesta & Oddi 1996; Jonsson *et al.* 2000; Frank & Jonsson 2003). In our architecture activities that have to be scheduled are organized as a network, where nodes represents activities and edges represents quantitative temporal constraints between them. Activities no longer represent a "blind static set" of entities that someone produced and gave to the scheduler, but they are continuously produced and scheduled at the same time by the planner. So, similarly to (Muscettola *et al.* 1992), planning here means to match specified patterns (the *domain theory* in a planning perspective) justifying the allocation over state variable or resources of some pre-defined activities required by the user as *goals* of the planning process.

This approach can be usefully applied to those domains where the scheduling problem is actually the hardest aspect, e.g., resource reasoning is very critical with respect to causal reasoning and requires specialized scheduling technologies. Those domains can be quite demanding for planners that integrate very generic scheduling features. Indeed this kind of domains often do not require strong planning capabilities. By enhancing a scheduling solver with some planning capabilities it is possible to tackle such problems in a quite flexible way, as we will discuss here with a practical example.

In this paper we describe how using a CSP perspective we have been able to (1) put causal reasoning into a scheduling framework, (2) model and (3) solve the corresponding planning and scheduling problems. We additionally present a sketchy description of OMP and describe its current solving abilities.

Planning as Resources Scheduling

To describe OMP we start from a scheduling framework which is able to manage temporal and resource constraints, and then try to understand how to increase the capabilities of this framework with planning features.

We model the integrated P&S environment as a set of sub-parts (or threads) that continuously evolve over time. On each thread several activities can be allocated. The evolutions of these sub-parts in each solution depends on which activities are allocated and when. For the sake of clarity we distinguish between two kind of sub-parts: *resources* and *state variables*.

Roughly speaking resources are the classical multi-capacity resources used in scheduling, where activities that specify a *consumption* between their start and end point have to be allocated, and the problem is to calculate when to start each activity in order to avoid over or under consumptions. The temporal evolution of these components in a solution is a piecewise constant integer functions of time computed by summing activities resource requirements at each instant.

State variables are components that assume states on a finite set over time. Several transition rules between these states are specified in order to constraint feasible evolutions for these components. Activities that have to be allocated over them specify a piecewise of temporal evolution stating which states the component can take between their start and end point. The problem consists in computing when to start each activity in order to avoid the overlapping of activities that require contradictory values. The temporal evolution of these components in a solution is a piecewise constant functions of time computed by intersecting state variable requirements at each instant.

The approach described here relies on a constraint-based representation for problems and aims at describing a framework where both planning and scheduling problem instances have as a common representation model, the Constraint Satisfaction Problem (CSP) (Tsang 1993). A CSP consists in 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 in 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 on 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.

Scheduling is one of the strong application areas of constraint programming. Several solid techniques have been developed to deal with several aspect of the problem (Baptiste, Le Pape, & Nuijten 2001). In our current effort we rely in particular to the *Precedence Constraints Posting* (PCP) approach (Cheng & Smith 1994; Cesta, Oddi, & Smith 2002) that is based on a temporal network where start and end points for each activity are mapped as time points. This approach allows the splitting of temporal reasoning from the resource usage management: using temporal information managed in a temporal database that can be updated and queried, the PCP approach builds *resource profiles* and synthesizes a set of additional *precedence constraints* between activities that, when posted, ensure resources are never over or under used. Moreover, because all activities (over both resources and state variables) in fact share the same temporal network, the integration of P&S is achieved by mixing temporal constraints in an environment where two specialized reasoners, a scheduler for resources and one for state variables, analyze the situation of a temporal constraint database, in fact posting constraints that affect the whole integrated problem. Hence our point of view: state variables and resources can be seen as *concurrent threads* in execution on a concurrent system, where a shared temporal model allows crossing relations between causal and resource usage aspects, even if two distinct reasoners affect them.

Figure 1 sketches the representation that is the core reference for problem solving. Three aspects are clearly distinguished: the temporal data base (the bottom layer), the problem model (in the middle layer) and the resource profiles (layer on the top) where resource profiles are draw as stepwise integer functions and the state variable profiles are

draw as ordered sequences of states.

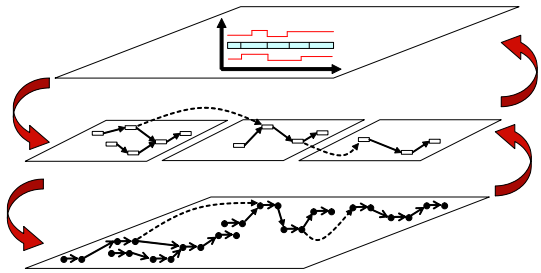

Figure 1: Resource and State-Variable Integration

It is worth spending few more words about the middle layer in the figure. Here it is shown how we represent the integrated P&S problem. Following a typical AI approach the planning problem is stated as a *domain theory* that represents which elements are interesting in the world we're working on and how things happen in this world, i.e. which rules the interesting elements must follow. As we saw we identify interesting elements as components that have temporal behaviors (resources and state variables) and we specify rules that these elements have to follow as patterns that specify which combinations of activities over these components are legal in our world. For instance if we are modeling an equipment that requires set-up activity before and after performing a certain activity, we specify that when this kind of activity is allocated, two *new* activities (automatically generated) have to be scheduled just before and after it. Thus starting from some goals stated as activities that must appear in the final schedule the pure planning problem can be seen as to build activity networks that match all specified patters, i.e. that contains all activities needed with temporal constraints among them. These networks are draw in the middle layer of the figure 1.

Moreover most complex domains patterns often require non trivial synchronization between different components: for instance when a state variable is performing a certain activity, it is easy that other state variable must perform other activities before, during or after it. These are represented in the figure as dashed cross lines between different networks (each network is supposed to be scheduled over a different component). When patterns involve activities both on state variable and resources we are actually linking the causal and the scheduling parts of the problem, for instance requiring a resource consumption when an activity is performed by an equipment modeled as a state variable.

Allowing these complex patterns it is easy to understand how activities in the networks can be partially ordered: we do not specify just sequencing constraints, but in general our integrated P&S problem can be seen as the problem of generating and scheduling partially ordered networks of activities starting from a user provided initial network of goals.

It is worth noticing that our model of the problem allow us to deal with resources directly using state of the art approaches to scheduling: in fact at each step of the computation the scheduling engines can analyze activity networks that have to be scheduled over resources looking for feasible ordering for activities, then providing to the planner several sets of precedence constraints that can solve the problem. Then we are putting causal knowledge directly into a scheduling engine, modeling the planning problem as a scheduling problem over both resources and state variables.

To demonstrate our approach let us consider an example from a space domain where a spacecraft has to achieve some goals with its payloads, like taking pictures with a camera or gathering some specific data with instruments. A simplified model for this domain, where observations have to be stored into an internal memory then downloaded to Earth, can be represented in our framework by using two resources, *Memory* and *Downlink_Channel*, a state variable *Satellite*, and a set of state variables $\{I_1, \ldots, I_n\}$ one for each on board instrument. The state variable *Satellite* should take values in the set $\{Locked(target), Locking(target), Unlocked()\}$ where $Locked(target)$ is its value when the satellite is pointed toward the object $target$, $Locking(target)$ when the satellite is moving to point its target and $Unlocked()$ is a sort of idle status. Each instrument I_i takes values $Perform\_Observation(target, data)$ when an observation of object $target$ is performed and information is stored in $data$, $Download\_Memory(data)$ when $data$ are downloaded to Earth (using communication channels) and $Idle()$, an idle status. Modeling this framework requires both planning and scheduling features: (a) a planning problem because performing observations requires the execution of some non trivial operations, such as pointing the satellite toward the object that has to be observed, *then* storing information *while* the satellite is locked, *then* downloading this information *when* Earth is visible from the satellite; (b) a quite hard scheduling problem, because usually satellites have limited resources, little memory and not so wide communication channels. Thus we need the right tools to deal with communication channels usage and memory management.

In order to model these causal constraints we must ensure that *when* a state variable I_i takes the value $Perform\_Observation(target, data)$, the state variable $Satellite$ is $Locked(target)$ and a value $Download(data)$ occurs sometimes *after* the observation is performed. Moreover, because we want to download data to Earth, when the value of a state variable I_i is $Download(Data)$, the state variable $Satellite$ must be $Locked(\text{“}Earth\text{”})$ (that can be for instance a dashed crossing line in figure 1). Finally the satellite must move toward its target before pointing it, thus each state $Locked(target)$ has to be immediately preceded by a $Locking(target)$ state. This is our domain theory, in which we specify how the application domain works by expressing a set of constraints over values that these variables can take over time. Activities that have to be scheduled over resources $Memory$ and $Downlink\_Channel$ are produced when an observation is performed (consuming memory) and when data are downloaded (freeing memory and using communication channels). Then we must require that when a state variable I_i takes a value $Perform\_Observation$ a resource consumption for resource $Memory$ has to be scheduled (another crossing line in figure 1 for instance),

while a production for the same resource and a consumption/production pair for the resource $Downlink\_Channel$ must be scheduled. Of course the entity of such resource usages depends on how many data have to be stored or downloaded. This model is explained more in detail in the next section. It is worth pointing out that, due to its ontology, because a state variable can take no more than a single value for each time instant, we are also modeling the fact that the satellite can be pointed only toward an object at a time, and if we want to observe more than one object or download data, we have to change its pointing status, in fact moving the physical object that the state variable $Satellite$ models.

From this point of view, once we specify a set of goals we want to achieve, as for instance in our example defining some operations as $Perform\_Observation(\text{“}Mars\text{”})$ with an instrument I_1 between two time points t_1 and t_2, in order to match all constraints specified in the domain theory, several other activities that have to be allocated over resources and state variables are automatically produced, generating activity networks such as those we saw before, that share the same temporal information. Thus, once all networks are generated, in fact unfolding the domain theory and ensuring all cause-effect relationships are guaranteed according to the domain theory, the problem becomes to find a feasible ordering for these activities in a way that each state variable takes at least and no more than one value at each instant of time and resources are never over used. This purpose can be achieved by scheduling them. Moreover propagation rules can be applied, in fact solving the whole problem as a CSP, alternating propagation and decision steps.

The Domain Theory

As we have seen before, we follow an ontological paradigm based on splitting the domain in several components, that can be state variables or resources, then specifying a domain theory as a set of constraints. From a general point of view, for each component in any “legal” behavior, when a value occurs, other values must also occur, matching a pattern that is specified from the user. Thus to specify a domain theory means to specify patterns that can be used by the planner to distinguish between “legal” and “non legal” behaviors for the system it is planning on. In a CSP environment the domain theory is an user defined set of constraints that have to be matched by each feasible solution besides constraints due to components ontology. Basically resources' ontology requires to avoid over or under consumption within the scheduling horizon, while state variables' ontology requires they must assume at least one and no more than one state at each time instant in the planning horizon. These “embedded” constraints are automatically enforced as precedence constraints posted by scheduling engines over activity networks. Thus when users specify a domain theory and a problem, they actually specify more constraints that have to be enforced in each feasible solution. In OMP each step in the planning and scheduling process is completely represented as a set of activity networks, one for each component, specifying which activities and in which order have to be allocated (or *executed*) over state variables or resources. These networks represent decisions that the solver took during the computation. Thus in OMP when an activity occurs in any position in one network, other activities must also occur in other parts of the instantiated domain model matching a pattern that is specified from the user in the domain theory. In fact, a domain theory in OMP is a collection of entries:

$$\begin{array}{ll} \langle ref\_act, & \langle constr\_act_1, temp\_rel_1\rangle, \\ & \langle constr\_act_2, temp\_rel_2\rangle, \\ & \vdots \\ & \langle constr\_act_n, temp\_rel_n\rangle\rangle \end{array}$$

Each entry says that when the activity ref_act appears anywhere in the networks, in order to obtain “legal” component behaviors, activities $constr\_act$ must also appear in the networks. These entries specify the causal links between constrained activities and the reference activity saying that they are in the final plan *because* the reference activity is in the plan, i.e. they *justify* the existence of the reference activity in the plan.

Moreover because OMP is also a planner able to manage quantitative temporal relations, for each entry, $temp_rel$ specify a sort of *offset* that is used to situate constrained activities in the right temporal position, by adding the offset to the temporal occurrence of the reference activity. These offset are specified as a combination of Allen's interval relations with metric constraints that can be expressed in continuous endpoint algebra (we will clarify that later with an example). The planner, in order to justify the presence of an activity in the plan, produces a sort of “*temporalized causal relationship*” between activities, that are mapped into edges in the networks.

The OMP domain theory is modeled using DDL.2 specifications. The DDL.2 domain description language (see (Cesta, Fratini, & Oddi 2004) for a more detailed description) is an evolution with respect to a previous proposal called DDL.1 (Cesta & Oddi 1996). The specification of a state variable component involves the definition of its possible states and its allowed state transitions. In DDL.2 a state variable definition specifies its name and a list of values, or *states* that the state variable may take. Each state is specified with its name and a list of static variable types. Indeed the possible state variable values for DDL.2 are a discrete list of predicate instances like $\mathcal{P}(x_1, \ldots, x_m)$. For each state variable $\mathcal{SV}_i$ we specify: (1) a name that uniquely indicate this kind of component; (2) a domain $\mathcal{DV}_i$ of predicates $\mathcal{P}(x_1, \ldots, x_m)$ and (3) a domain $\mathcal{DX}_j$ for each static variable x_j in the predicate. In order to describe a resource component it is enough to specify its name and some parameters about its consumption boundaries.

In figure 2(a) we show a possible DDL.2 model for the component $Satellite$ described in the last section where the type $TARGET$ takes values on a set containing labels for each interesting object in the world, plus, of course, a label for the Earth (while $target$ is an element of this domain). For each state we specify which states can follow it and which state can hold before it, expressing in that way which state transition rules are legal for that component. Value $Unlocked()$ for instance can hold at least for 1 second and there is not upper limit to the time the satellite can be in this idle status (statement $[1, +INF]$), and can be followed

(Statement MEETS) by a value $Locking(target)$. Similarly MET-BY statement allows to specify which value the component can take just after $Unlocked()$ value. Roughly speaking this model describes a simple component which behavior is an alternation of sequences $\ldots Unlocked() \rightarrow Locking(target) \rightarrow Locked(target) \rightarrow Unlocked() \ldots$ and so on.

```
SV Satellite (Unlocked(),Locking(TARGET),Locked(TARGET))
{
COMP
{
STATE Unlocked() [1, +INF] {      MEETS { Locking (x:TARGET); }
                                  MET-BY { Locked (y:TARGET);}}
STATE Locking(x) [1, +INF] {      MEETS { Locked (x)}
                                  MET-BY {(Unlocked())}}
STATE Locked(x) [1, +INF] {       MEETS { Unlocked ()}
                                  MET-BY { Locking (x)}}
}}
```

(a) DDL.2 State Variable description

```
COMP
{
STATE Perform_Observation (target,data) [time(target) , time(target)]

SYNC
{
DURING Satellite Locked(target) [5,+INF] [5,+INF];
BEFORE Instrument Download_Memory(data) [time(data) ,time(data)];
USE Memory memoryOcc(data) [0, 0] [0, 0] AFTEREND;
USE Channel memoryOcc(data) [0, 0] [0, 0] FROMSTARTTOEND;
}
}
```

(b) DDL.2 Compatibility

Figure 2: DDL.2 specifications

The main interesting feature of this language is the description of the so called *compatibilities*, a sort of schema used to express domain theory constraints in a compact way. Basically each compatibility expresses a set of patterns as an AND/OR tree of activities. The root is the reference activity, and the tree can be "unfolded" by the planner to generate several patterns that can be used to justify the reference[1].

In our example some compatibilities have to be expressed for instrument components. When they take the value *Perform_Observation(target,data*) we need that (1) $target$ must be visible (then a synchronization is required with a $Locked(target)$ value for state variable *Satellite*; (2) observed data have to be downloaded (then it must exists a following state *Download_Memory(Data)* in the same component behavior). It is worth noting that the language allows to specify very complex constraints, like for example the need to make data downloadable not before a certain slack of time (for example, to perform some on-board elaboration on them) and not after another slack of time (in order to avoid information starving); (3) you must have enough free memory and enough channel rate to perform your operations, then activities over *Memory* and *Channel* resource must be allocated. The amount of resource required from activities depends on how much data the instrument produce or it's able to transmit in time. Similarly for *Download_Memory* state some compatibilities have to be expressed, to be sure that only previous retrieved data can be downloaded and Earth is visible. In figure 2(b) there is a DDL.2 compatibility specification for a generic instrument[2]. It is easy in this figure to understand how temporal relationships are expressed using Allen's relationships (as $DURING$ or $BEFORE$ for instance) integrated with metric constraints: for instance $DURING\ Satellite\ Locked(target)\ [5, +INF]\ [5, +INF]$ means that in the activities network for the state variable $Satellite$ must appear an activity $Locked(target)$ that must start at least 5 time units before the start of the reference activity $Perform\_Observation$ (there is not upper bound) and must finish at least 5 time units later (this meaning is implicit in the statement $DURING$). In a few word the satellite must be locked at least 5 seconds before and after the instrument takes an observation.

Once you modeled the domain theory you can express the problem that have to be solved specifying some goals like actions that have to be performed by instruments components, as temporally situate *Perform_Observation* values, and/or a set of pre-defined activities that have to be allocated over resources in any feasible solution (goals over resources). The solutions produced are feasible both from planning point of view (meaning that downloads follow observations and every operation is performed when the satellite is right oriented) and from the scheduling point of view (memory and channel are never overused).

It is worth pointing now how, due to their ontology, we use state variables to model and reason about the planning side of the integrated problem, while we use resource components to deal with the scheduling side. In our architecture it is feasible to deal also either with a "pure" planning problem or a "pure" scheduling problem. In a "pure" planning domain there are only state variables, while a "pure" scheduling problem can be modeled using only resource components. State variables are able to represent causal relationships as very common in many engineering applications, where finite state machine are used to model components' dynamical properties. On the other hand we do not need so many explanations about how resource components can be used to deal with a classical scheduling problem[3].

A further aspect worth remarking is that the explicit representation of resources in this approach implements a concept of "bidirectionality" between planning and scheduling that is peculiar of this architecture.

Problem Solving

We have shown how in our framework we dynamically generate networks of activities that have to be allocated over state variables or resources. These activities share a common temporal model, and are linked to each other via temporal constraints. Thus we can reason on each of these networks, deducing, as typical in CSP problem solving, necessary constraints via propagation, and analyzing the under-

[1]The semantics of compatibilities is consistent with that in HSTS (Muscettola *et al.* 1992; Muscettola 1994) and related works.

[2]For the sake of brevity we do not show the MEET and MET-BY statements. *time* and *memoryOcc* are two integer functions. The first computes the time needed for a payload operation, the second the amount of data it produces.

[3]In fact, OMP is able to solve RCPSP/max problems with a search strategy that integrates a complete search with a conflict analysis based on the ideas in (Cesta, Oddi, & Smith 2002).

lying temporal problem in order to compute which precedence constraints have to be posted to guarantee a feasible allocation of these activities over components. All temporal constraints deduced via propagation and all precedence constraints posted as search decisions affect the whole shared temporal representation, and each decision has an automatic feedback on the temporal position of all other activities.

In the pure scheduling case methods to deduce *necessary* constraints by propagating information about resource usages have been widely studied. It is possible do deduce a set of ordering constraints between activities that must be posted, just because otherwise a resource violation surely occurs. We studied how to work over state variables, implementing a method for discovering an inconsistency of posted constraints (i.e. a temporal interval where the intersection of values allowed by constraints over that interval is an empty set) and a propagation algorithm which is able to discover new necessary orderings between tasks in order to avoid necessary inconsistences over state variables (some details about these aspects are provided in (Fratini, Cesta, & Oddi 2005)). In most cases this is not enough in order to solve the problem. Sometime a search decision is necessary, even when we know that two or more activities can not overlap but, analyzing the underlying temporal problem, we are not able to calculate any necessary precedence constraint between activities, more orderings are temporally feasible. Thus a search decision step is needed, basically among the temporally feasible orderings for activities.

Every time that an activity is added to a network a *propagation step* is fired that is actually out of the planner control: constraints posted during this step are necessary, i.e. they prune only non-feasible solutions, meaning that *any* feasible solution *must* contain these constraints. So it is not necessary any kind of active control, this process starts automatically every time that the planner put an activity into a network, and it is performed by *manager modules* showed in figure 3. With the *Temporal Network* these modules constitute a *constraint database* that maintains information about the current partial solution and propagates decision effects pruning the search space (we remind here also that in OMP the temporal problem is represented as an STP, that can be solved only via resource propagation, without any kind of search (Dechter, Meiri, & Pearl 1991)).

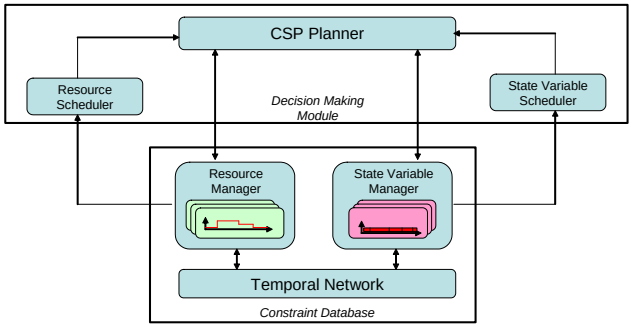

Figure 3: OMP Software Architecture

On the other hand scheduling precedence constraints are *search* decisions, thus they *could* cut some feasible solutions. Thus planner is in charge to choose between different orderings, and it must keep track about these decisions. In OMP scheduling modules play the role of constraint databases analyzers, in charge of inspecting networks status and provide information to the planner about what can be do in order to ensure state variables and resource are correctly used (meaning that no over or under consumption occur on resources and no contradictory states are required over state variables). Due to their decision support role, these modules are shown in figure 3. Together with the Planner they form the *Decision Making* part of the architecture.

The problem solving algorithm, starting from partially ordered networks of activities that must appear in the final solution (user goals), try to complete them ensuring that all activities in the solution are justified (with respect to domain theory patterns) and any violation arise on resource usage. There are two types of flaw that can arise at each step of the computation: (1) an unjustified activity and (2) a violation on resource or state variable usage. In the first case the planner has to analyze the domain theory and choose a pattern he can use to justify the action. In the latter case the decision is among different orderings for activities (if they exist!) provided by the schedulers.

It is worth pointing out that the problem solving algorithm, as typical in CSP, is an alternation of information retrieval from the constraint database and new constraints posting over the same database (search decisions). When the planner has to justify an action for instance, it can easily check if it exists some activities previously inserted that can be used (meaning that matches the logical requirements and that can be forced to occur in the right temporal interval). In this case it is not needed to insert new activities, and temporal constraints that link the needed activity with the previous inserted are enough. If there are no activities that can be used, the planner needs to insert new activities, so the process of expansion, typical of planning, restarts because the new activity have to be justified.

It is interesting to underline the fact that it is easy to explore different degree of integration between planning and scheduling. In fact at each step the planner must decide between causal expansion (to justify action) and scheduling (if any violation arise). The ratio between how many expansion step the planner performs with respect to scheduling steps determines the interleaving degree. If it tries to justify everything without taking care of resource violation, and scheduling everything only when anything is justified means in fact to serialize planning and scheduling, while you can have a complete interleaved approach alternating expansion and scheduling at each step.

Related Works

The synthesis of OMP is quite recent and as a consequence this solver cannot be considered a stable software system like other similar proposals. Nevertheless it is worth making a comparison with other systems that, for different reasons, have most in common with our planner.

Visopt ShopFloor (Bartak 2002) is grounded on the same idea as OMP: 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 described using resources with complex behaviors. These resources are close to our state variable, but they are managed using global constraints instead of a precedence constraint posting approach as we are currently doing. As we said the approach we are pursuing allow us to deal with non trivial multi resource scheduling problems, solved in OMP directly integrating in the architecture several state of the art results in scheduling research. Moreover, although we are working on P&S integration, we maintained a clearer distinction, with respect to Visopt, between planning and scheduling in problem modeling features, allowing users to describe an integrated domain theory and to state a set of goals that the planner must achieve.

IxTeT (Ghallab & Laruelle 1994) follows a domain representation philosophy based on state attributes which assume values on a domain. Moreover it represents system dynamics with a STRIPS-like logical formalism. In contrast with OMP, resource reasoning is used as a conflict analyzer on top of the plan representation, while integrating large resource propagation strategies within its basic representation is not easy and we believe that a way for integrating complex propagation of resources is with a separate module similarly to what is done in OMP.

HSTS (Muscettola 1994), has been the first to propose a modeling language with explicit representation of state variables. In fact we extended an HSTS-like state variables modeling language with multi-capacity and consumable resources. A clear difference is that in our approach we reason about these resources in a separate module while RAX-PS views resources as specialized state variables. Their view is certainly appealing and, to some extent, formally clean but unfortunately the problem of integrating in a clean way multi-capacity resources is far from being solved. In fact, while it is immediate to represent binary resources as state variables, it is very difficult to model and to handle with several aspects, common in many complex domains, involving non trivial counts about resource usages: we think that in these cases the best way is to exploit state of the art scheduling technologies.

Finally it is worth noting that when resource contention is a critical issue it is difficult to plan without a direct *quantitative* feedback about resource status in order to prune the search space "while" the planner is looking for a solution. This sort of pruning is potentially more effective with respect to the trace of resource status with counters checked during planning to understand if any violation arises as done in some of the domain independent planners. An interesting alternative is to use a scheduler that can *drive* the planner during its work.

Thus we think that in integrating planning and scheduling, both problem solving abilities have to be "preserved", meaning that also native scheduling engine have to be exploited and resources cannot be handled "simulating" them.

Conclusions

This paper describes our current attempt at integrating planning into scheduling in a CSP approach. We have described the basic motivations which brought us to develop OMP, a software architecture that integrates a constraint based scheduler with the ability to synthesize new activities. The synthesis of new activities is performed by reasoning on planning knowledge. A strong motivation for this effort is to maintain state-of-the-art scheduling capabilities in problems where resource reasoning is a main issue.

We have followed the idea of integrating planning and scheduling problems by rationally blending in the same architecture various state-of-the-art CSP algorithms. We have shown how modeling the integrated planning and scheduling problem as concurrent evolving components allows us see the all approach as network scheduling, where networks are automatically generated in the same architecture from a compact domain theory description and a set of goals that have to be achieved.

The integrated approach was built over a shared CSP representation of all the relevant information: time, resources and causal knowledge. Thus state of the art propagation and scheduling algorithms are embedded into the architecture. Thanks to our effort in exploiting similarities and differences between planning and scheduling we were able to present a framework where causal knowledge modeling and management are very close to a pure scheduling problem, which in turn has allowed us to follow an approach similar to that of task scheduling.

Acknowledgments

Authors would like to thank Angelo Oddi for common work on this topic and the other members of Planning and Scheduling Team [PST] at ISTC-CNR for creating the stimulating environment in which this research is developed.

References

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

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

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

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

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

Cheng, C., and Smith, S. 1994. Generating Feasible Schedules under Complex Metric Constraints. In *Proceedings 12th National Conference on AI (AAAI-94)*.

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

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

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

Frank, J., and Jonsson, A. 2003. Constraint based attribute and interval planning. *Journal of Constraints* 8(4):339–364. Special Issue on Planning.

Fratini, S.; Cesta, A.; and Oddi, A. 2005. Extending a Scheduler with Causal Reasoning: a CSP Approach. In *Proceedings of the Workshop on Constraint Programming for Planning and Scheduling (ICAPS'05)*.

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

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

Laborie, P., and Ghallab, M. 1995. Planning with Sharable Resource Constraints. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*.

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

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

Pecora, F., and Cesta, A. 2005. Evaluating plans through restrictiveness and resource strength. In *Proceedings of the 2nd Workshop on Integrating Planning into Scheduling (WIPIS), AAAI-05, Pittsburgh, PA, USA*.

Srivastava, B.; Kambhampati, S.; and Do, M. B. 2001. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence* 131(1-2):73–134.

Tsang, E. 1993. *Foundation of Constraint Satisfaction*. London and San Diego, CA: Academic Press.