Scheduling in a Planning Environment

A. Garrido, M. A. Salido, F. Barber

Dpto. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, Camino de Vera s/n 46071
Valencia, Spain
{agarridot, msalido, fbarber}@dsic.upv.es

Abstract. In a real planning problem, there exists a set of constraints (both temporal constraints and resource usage constraints) which must be satisfied in order to obtain a feasible plan. This requires a scheduling process (after the planning process) which should guarantee the availability of resources and the satisfiability of all the problem constraints. Several approaches have been proposed to deal with planning and scheduling problems. However, these approaches have drawbacks which will be presented here. This paper deals with the main features of a scheduling process in an integrated architecture of planning and scheduling, where both processes work in a simultaneous way. Thus, the executability of each plan is guaranteed as it is being obtained by the planner. The planning process searches among alternative partial plans, where each one of them has its own ordering relations among actions, resource requirements, intermediate states, etc. Since these constraints are provided while the plan is being obtained, the proposed scheduling process should be able to manage them as they are being known. Thus, the scheduler should not obtain a solution after each new asserted constraint but rather it should only maintain the consistency among all the asserted constraints. In addition, the planner keeps track of several alternative open plans, which are suitable for being expanded in each moment. For this reason, the scheduler should maintain the effects of the constraints belonging to different plans that are being explored by the planner. Hence, both specific planning and scheduling optimisation criteria are used in order to improve the behaviour of the integrated system, its efficiency and the quality of the obtained plan.

1. INTRODUCTION

In a planning problem, actions usually require use of shared resources in order to be executed. Moreover, several temporal constraints should be satisfied during the plan execution: action durations, effect persistences, temporal constraints on problem states, due times, etc. In usual planning processes, resource usage and satisfiability of problem temporal constraints are not considered. Thus, planning systems obtain a plan as a partial or total ordered sequence of actions, and a later scheduler process should check the feasibility of the plan according to the available resources and problem constraints. Therefore, a correct plan may not be executable due to violation of some temporal constraint or unavailability of shared resources. Thus, a new plan should be obtained and there will be a loss of system performance.

On the other hand, temporal planners can reason about metric constraints such as parcPLAN (El-Kholy 1996) and IxTeT (Ghallab and Laruelle 1994). These temporal planners deal with temporal data by means of an explicit representation of time managing qualitative (ordering relations commonly used in planning) and quantitative constraints (release times and durations used in scheduling). In a more integrated way, there are planning systems such as Tosca (Beck 1993) and O-Plan (Beck and Tate 1995, Currie and Tate 1991) which integrate both planning and scheduling processes in a single system. However, a drawback appears in these cases: it becomes difficult to determine when the system is planning or scheduling: "*it is easy to see that O-Plan works, but it is difficult to see why*" (Bäckström 1998). Since a specific process of planning or scheduling does not exist, it becomes difficult to determine certain optimisation criteria (which, moreover, can be integrated in a central module). Consequently, we agree planning and scheduling integration is necessary and these two processes should be performed simultaneously. However, we think these processes are different enough to be distinguished during their execution in the integrated system. Planning processes deal with what actions are going to be executed whereas scheduling processes deal with when these actions are going to be executed (Dean, Greenwald and Kaelbling 1994). In another way, planning implies reasoning about actions and system states, and scheduling implies reasoning about actions, resources and time (Verfaillie, de Givry and Lesaint 1999). Therefore, it is not irrelevant to study both processes in a separate way in order to improve finally the performance of the integrated system (Bäckström 1998). Thus, an integrated system may obtain many benefits from both the planner and the scheduler, such as utilisation of shared heuristics, decrease the search space, performance improvements, etc. (Laliberty et al. 1996). Nevertheless, this integration usually is not quite frequent due to the fact that it is neither easy nor intuitive: "*these systems do not integrate well*" (Smith, Lassila and Becker 1996).

This paper deals with the main features of a scheduling process in an integrated architecture for planning and scheduling (Garrido et al. 1999). In our system, the planner and the scheduler work simultaneously in an integrated way. Here, the scheduler guarantees the satisfiability of temporal constraints and resource availability for each partial plan as these plans are obtained by the planner. This way, the system recognises the invalid plan and this plan is immediately discarded. Furthermore, even though the plan

is executable it may not be efficient enough or optimal. For this reason an alternative plan may be needed with the objective of reducing its cost.

One of the main features of our scheduling approach is its interactive behaviour and its independence from the planning system. This approach is valid for every planning system, both forward and backward chaining planners. The scheduler allows the integrated system to prune partial plans, avoiding the generation of invalid plans. Furthermore, the scheduler is able to manage several partial plans, which have been generated by the planner. Another important feature is the use of heuristics, characteristics of the scheduler, which will speed up the integrated process and improve its behaviour.

In this section we have presented the introduction to this paper. We propose the integrated system, its main features such as problem specification language (to model the problems), and its architecture in section 2. The scheduling process, its behaviour through an example and the way we manage all the temporal constraints and resource availability are described in section 3. Conclusions are discussed in section 4.

2. THE INTEGRATED SYSTEM

In this section we expose a high-level general view of our integrated system (Fig. 1). The domain representation is obtained from the problem domain by means of the specification language. The domain representation consists of the problem objects (including the resources), the actions and the problem constraints. The integrated system of planning and scheduling solves the problem in order to achieve the executable plan. Nevertheless, new problem constraints or incidences may appear during the execution period. In this case, a reactivity stage is needed to obtain a new optimal plan according to the new problem constraints. All these elements are detailed below.

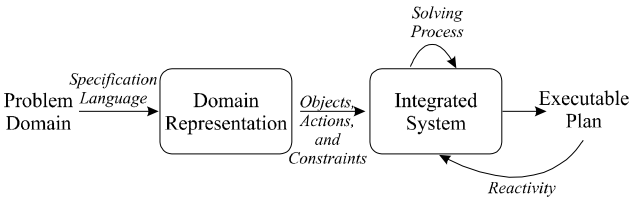

Fig. 1. General view of the integrated system (from Garrido et al. 1999)

2.1. Problem Specification Language

In order to model and analyse the problem domain, we use a specification language, which allows the user to define the next elements:

- Domain object hierarchy. Classic approaches in planning use a declarative-language by means of first-order predicates for domain description (Penberthy and Weld 1992). In contrast to these schemes, we always maintain the same structure for literals (Garrido et al. 1999):

  <*class-name*> <*object*> <*slot-name*> <*value*>

  This frame-based structure allows us to model real application environments. The object hierarchy can represent problem objects as well as the resource hierarchy. There exists a special class for shared resources as in (Smith, Lassila and Becker 1996). The resources are shared albeit nonsimultaneously by the actions. If objects are resources there are several slots by default, such as *quantity* (number of items), *resource availability* (temporal constraints that indicate when the resource can be used), *service* time (how long the resource is used by default), etc. Moreover, the user can also define the initial situation and goals to achieve by using the previous structure.
- Actions. Actions can be primitive actions, which cannot be divided any further or macro-actions, which group primitive actions in an established partial or total order. We can refine every macro-action in its primitive actions carrying out the planning and scheduling process through a hierarchy of different levels. Thus, we can obtain an initial plan that will be detailed in following steps of the process by means of a refinement method (Dean, Greenwald and Kaelbling 1994).
- Problem constraints. These constraints can be applied to different elements of the problem:
  - Temporal constraints over the entire plan. They indicate the possible execution duration of the plan by means of the possible beginning and ending of the plan.
  - Constraints over the resource usage. Due to the fact that resources cannot be simultaneously used by more than one action, constraints must guarantee that actions do not use the same resource in the same time.
  - Constraints over the ordering of actions because there are actions that must be executed in a specific order.
  - Constraints over the action durations. These durations can be dependent of the order in which they are executed.
  - Temporal constraints over the problem objects and their states (attributes). For instance, an object cannot be held in the same state for more/less than a determined interval, an object must reach a specific state in a determined moment, etc.

2.2. Architecture of the Integrated System

The planning and scheduling modules in the integrated system (Fig. 2) share data structures of the system with the common information. This shared common information is stored in a special kind of data structure similar to a blackboard model (Laliberty et al. 1996). The planner must accede to each data related to actions, their ordering, initial situation and goals. On the other hand, the scheduler must keep all the information related to allocation and nonsimultaneous resource usage, temporal constraints and ordering among actions.

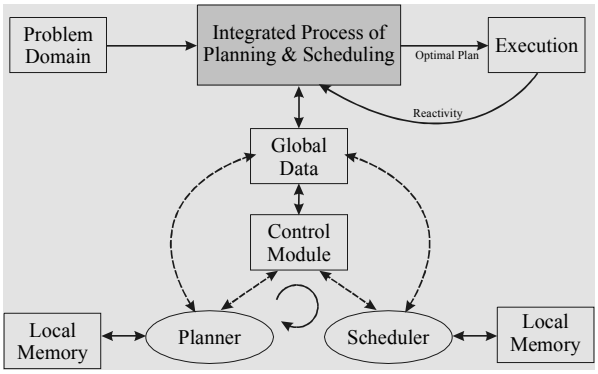

Fig. 2. Integrated architecture of planning and scheduling

In the integrated system, a strong communication should be carried out during the construction process of each partial plan. The planning and scheduling processes exchange information, by means of the shared data structures, in order to obtain a more efficient integration. Communication between the planner and the scheduler must occur:

i) Every time a new planned action (or an existing one) solves a precondition. In this case, the scheduler must update the ordering among the actions (and the ordering of the resources which are used by these actions) according to the new causal-link.
ii) When the planner demands a resource that must be used in an action. Here, the scheduler must update the sequence of utilisation of the resources in order to avoid a simultaneous usage.
iii) When a new ordering among actions is established due to a planning conflict resolution. As in the first case, the scheduler must update the ordering among the actions.

In all these cases, the planner must inform the scheduler about:

- The action to be planned. Since the planner works on various excluding alternative plans (Fig. 3), the planner must specify in which alternative plan the action is used. Each alternative plan represents a plan with different actions to achieve the problem objective. Only resources in actions of a same partial plan must not be simultaneously used. Therefore, the scheduler must be able to manage the possible excluding alternative plans at the same time, which are shaded in Fig. 3.
- The resource list to be used in this action. The planner may know the resource list to use due to another previously planned action in the same partial plan or because the planner requires some specific resources. In this case, the scheduler must not allocate new resources, but it will check if this allocation is consistent with the known constraints for the required resource in its partial plan. If the planner does not know the resource to be used, the scheduler should assign the necessary resources for the action, in accordance with the action specification, by using optimization criteria.
- The order of the action to be planned. The planner may establish an order, both partial and total, among several actions of each partial plan. This order is related to other actions involved in its plan. This mechanism permits us to establish some ordering criteria by indicating that one action precedes another. If this ordering is the result of solving an ordering conflict, the scheduler must reorder the actions and determine if the new ordering is consistent according to the existing constraints. For instance, if an ordering is not feasible because of the violation of any temporal constraint or because there is not any available resource, this plan will be discarded.

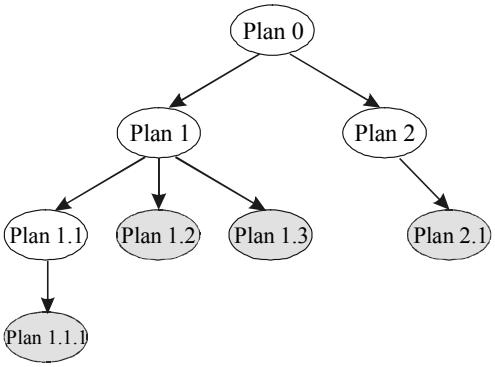

Fig. 3. Search among alternative plans (current feasible partial plans are shaded)

When the scheduler detects an inconsistency, it communicates to the planner to discard this action. Furthermore, the scheduler recommends the planner a list of alternative available resources. In addition, the scheduler could suggest which partial plan (from the frontier of the plan tree (Fig. 3)) should be expanded, taking into account which partial plan is less constrained or imposes less constraints over resources. This feature can be a valuable heuristic for improving the planning process. Hence, a great level of integration is required between the planning and scheduling processes. For instance, let be P_0 the current partial plan obtained as a result of a search in the space of partial plans (Fig. 4). This plan is expanded by adding new actions or steps that will achieve the final objective. Several alternative plans P_a, P_b and P_c might be generated, which introduce new steps S_a, S_b and S_c, respectively. If the plan P_b is selected, the scheduler must check that its temporal constraints are satisfied and allocate the resources (only if it is necessary). In order to guarantee the constraints of a plan, the scheduler may constrain the partial sequence of steps of that plan. Therefore, the plan P_b' is the plan P_b with all its constraints satisfied. Next, the plan P_b' might be expanded, the plans P_{b1}', P_{b2}' and P_{b3}' would be generated and the process would continue until accomplishing all the problem objectives. Finally, the integrated system obtains an executable (and eventually optimal) plan according to the resource usage and other optimization criteria.

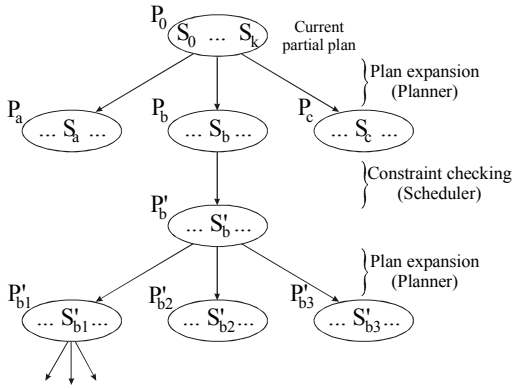

Fig. 4. Process of plan expansion and constraint checking by the planner and the scheduler, respectively

During the execution time, some incidences might appear (for instance if some resource becomes unavailable). In this case, a new reactivity process to repair all the conflicts would be necessary to obtain a reassignment of resources, if possible. Otherwise, the plan should be modified to accomplish the new problem requirements. This gives rise to a rescheduling or replanning problem, which is solved by means of a repair method (Dean, Greenwald and Kaelbling 1994).

As a result of the integration, we can obtain the following advantages:

- We can detect an inconsistency (due to resource unavailability, temporal constraint violation, etc.) in a partial plan as soon as this inconsistency appears. Since this inconsistency implies a nonfeasible partial plan, this plan is quickly discarded. Hence, the efficiency of the global process is improved.
- Since the system uses a specific module of scheduling, the system can manage more complex constraints than temporal planners. Moreover, we can define optimisation criteria (both in the planner and the scheduler) which improve both the efficiency of obtaining the plan and the quality (optimality) of the obtained plan.
- According to the way of obtaining the partial plans (Fig. 4), the final plan is executable.

3. THE SCHEDULING PROCESS

Once established this general architecture, we will analyse the requirements that the scheduling process requires. On one hand, one of the main aims of the scheduling process is to guarantee the executability of the achieved plan according to available resources, context constraints, etc. (Dorn and Froeschl 1993). All temporal constraints in the problem must be satisfied. Furthermore, the shared resource usage must also be consistent; i.e. enough available resources must exist to ensure the fulfilment of each planned action when it is finally executed. On the other hand, another aim is to guarantee the optimality of the obtained plan, according to its cost, due times and some optimization criteria.

In order to carry out the integration defined in the previous section we need a scheduler with a special behaviour, which should be more dynamic and interactive than traditional scheduling processes. Traditional schedulers are based on *Constraint Satisfaction Problems* or *CSP*s (Kumar 1992, Sadeh and Fox 1996). These schedulers are not directly applicable here, because of its lack of flexibility: it is very costly to have to obtain a new solution every time a new constraint is added or eliminated. For each new set of constraints (which are the result of including or excluding constraints), a CSP process must resolve the entire problem in order to obtain a new solution. However, when the set of constraints is modified, the previous solution may become invalid. It is clear that incremental CSP methods might be used here, but we do not need the solution to the problem in each step (Tsang 1993). We only need to assume consistency at each new asserted constraint. Furthermore, the scheduler must be *contextual*, i.e., it must be able to manage several alternative plans with complex temporal constraints simultaneously.

Main features of our scheduler, its behaviour through an example and the way of managing the temporal constraints are detailed in this section.

3.1. General considerations

Temporal constraints in our problem can be represented by a temporal network where nodes represent time points and arcs between them represent disjunctive metric-temporal constraints (Dechter, Meiri and Pearl 1991). Working with disjunctive metric-temporal constraints is a complex task because of it implies working with a huge number of equivalent networks (one for each disjunction). For instance, in a typical problem of scheduling, if the number of tasks on a common shared resource is n, the number of equivalent different nondisjunctive graphs for each generated graph will be 2^n (Salido, Garrido and Barber 2000).

Our scheduler uses a module for reasoning with temporal constraints, the *Temporal Constraint Network Manager* (*TCNM*). The TCNM guarantees the consistency of the temporal network by using a closure process, which propagates each new constraint to all nodes of the network (Barber 2000).

In traditional scheduling systems, the entire set of problem constraints is known in advance, so the aim of a scheduling process is to obtain a solution which satisfies these constraints. Here, *CSP* techniques are usually used (Kumar 1992, Sadeh and Fox 1996). Alternatively, other schedulers work on a set of initial solutions which may not satisfy the problem constraints, and the schedulers repair them over time (Dean, Greenwald and Kaelbling 1994, Smith, Lassila and Becker 1996).

In opposition to these traditional scheduling processes, the problem constraints are incrementally supplied, in our case, by the planner while each partial plan is being generated. At each new constraint, the scheduler guarantees the consistency of all the currently known constraints in each partial plan. Our scheduler works in a progressive way. As in (Boddy 1993), schedules are constructed by a process of iterative refinement. We believe this approach to be more flexible because it allows us to add constraints in a dynamic way. When a new constraint is added into the system, the

scheduler will detail the schedule constraining the domain of possible values. The scheduler does not obtain the solution that satisfies all the current constraints after each constraint is asserted, but maintains all the minimal sets of values that might be solutions. An inconsistency is produced when the domain of possible values becomes empty by the effects of a constraint C_i. For instance, we will see an example of two actions which use the same nonshared resource in Fig. 5.

Let A_x and A_y be two actions that can be executed in any order of precedence and that use the same resource. Therefore, A_x must be executed before A_y (a) or vice versa (b). At this moment, the scheduler only maintains the interval of possible solutions in the timeline but without allocating any concrete value as in traditional *CSP*s. Next, if a constraint indicates that A_x must be executed before A_y, the scheduler will discard the established order "A_y before A_x". With this information, if a new constraint asserts the fact "A_y before A_x" this constraint will be treated as an inconsistency and it will be discarded. Furthermore, the scheduler may impose more restrictive constraints: it may impose additional constraints on plans due to temporal or resource usage constraints.

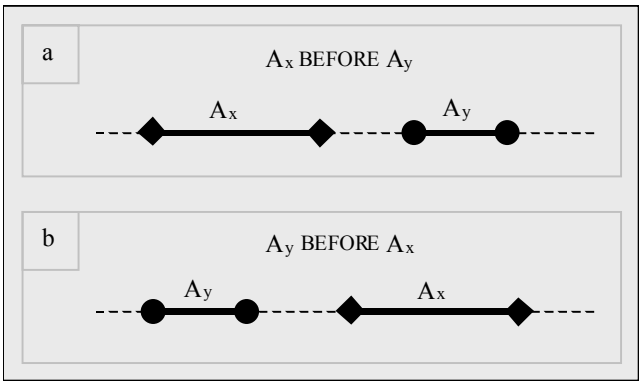

Fig. 5. Example of two actions A_x and A_y which use a nonshared resource

3.2. Scheduler behaviour

The scheduler must represent the necessary information in its temporal graph for each new planned action. Every action of each plan is translated into one or more temporal constraints, which are managed by the TCNM. The consistency can be guaranteed thanks to this TCNM which checks each new constraint introduced into the scheduling system.

Following, let us see the three levels of temporal constraints that arise in this approach.

Temporal Constraints in the Action Level

At this level, constraints are based on durations of actions, ordering relations and constraints among their time points.

For every action (A_i), the scheduler must create two new time points, which represent the start time point ($A_i.on$) and the end time point ($A_i.off$) of the respective action. The disjunctive durations are represented by $A_i.on$ *{(*$dmin_1$ $dmax_1$*), (*$dmin_2$ $dmax_2$*), ..., (*$dmin_n$ $dmax_n$*)}* $A_i.off$. Each interval represents a different disjunction in the duration of that action. Furthermore, the scheduler will check the ordering constraints with other actions. For instance, if there is another action A_j that must be executed before A_i, the constraint $A_j.off$ *{(0 ∞)}* $A_i.on$ will have to be satisfied. On the other hand, if an action A_j must be executed before a time point TP_j, it is represented by $A_j.off$ *{(0 ∞)}* TP_j.

Temporal Constraints in the Resource Level

Here, constraints are based on durations and nonintersection of shared resources due to their nonsimultaneous usage. In addition, constraints among actions and time points of resource usage are included.

For every resource (R_x) used in an action (A_i), the scheduler must create two time points. They represent the start time point ($R_xA_i.on$) when the resource in that action may start to be used and the end time point ($R_xA_i.off$) when the resource in that action may have finished being used. The durations (resource usage) are represented by disjunctive intervals between the beginning and the ending of the resource usage: $R_xA_i.on$ *{(*$dmin_1$ $dmax_1$*), (*$dmin_2$ $dmax_2$*), ..., (*$dmin_n$ $dmax_n$*)}* $R_xA_i.off$. The beginning and the ending of the resource utilisation is related to the start and end time points of the action they belong to. Hence, a resource may be used during all the action or only during a part of it. There may be an offset, either positive or negative, between the start point (end point) of the action and the start point (end point) of the resource in that action. Besides, it is necessary to guarantee that the use of a resource R_x in an action A_i (represented by R_xA_i) is not simultaneous with the use of the same resource in another action A_j (represented by R_xA_j). It can be performed simply by indicating the usage of the resource R_x in A_i is before or after the usage of that resource in A_j.

Temporal Constraints in the Attribute Level

At this level, the constraints between the object's states (attributes) are represented. Constraints appear due to relations between states of one or more dynamic attributes. For instance, an attribute cannot change its value during a time window.

For every pair state-attribute (St_kAtt_x), the scheduler will create two new time points, which represent the beginning ($St_kAtt_x.on$) and the ending ($St_kAtt_x.off$) of an attribute state. In a similar way, the duration of these attribute states can be represented as $St_kAtt_x.on$ *{(*$dmin_1$ $dmax_1$*), (*$dmin_2$ $dmax_2$*), ..., (*$dmin_n$ $dmax_n$*)}* $St_kAtt_x.off$. If there are some ordering relations among states, we use the type of constraints $St_kAtt_x.off$ *{(0 ∞)}* $St_lAtt_x.on$ if the attribute is the same. If the attributes are different, we use the constraint $St_kAtt_x.off$ *{(0 ∞)}* $St_lAtt_y.on$.

Since these changes in attribute states are produced by the effects of some actions, each pair state-attribute must be related to the action that produced its change. As in the resource level, there may be a delay, either positive or negative, between the beginning or ending of the action, and the change of the attribute state. The notion of persistence can be represented by a delay (see Fig. 6) in the ending of the state change (if the persistence is positive the value $St_kAtt_x.off$ will be later than the ending of the action, which produces the change in this attribute state).

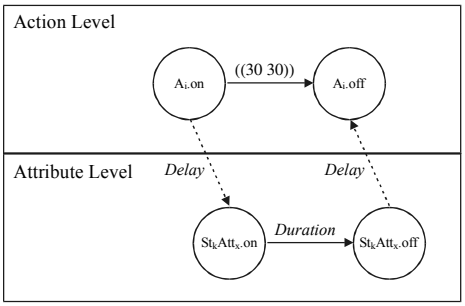

Fig. 6. Representing the persistence in the attribute level

As we can observe from above, the three levels are quite similar. In fact, the steps to carry out these levels are practically the same: creating the time points, establishing the duration and managing the precedence relations.

Moreover, it is important to realise these three levels are repeated in each alternative plan managed by both the planner and the scheduler. The time points of different plans must not be related because they represent distinct alternatives to achieve the final plan. If some inconsistency is produced in a determined plan, the planner will discard that plan and the scheduler will discard the graph with the three levels of that plan.

3.3. Illustration through a simple example

In order to illustrate better the previous levels of temporal constraint management, we show a simple example of how the actions planned by the planner are managed by the scheduler.

We will use as example a modified version of the well-known ferry problem (Barret et al. 1996). The objective is to transport some vehicles from the margin of a river to the other one. In our case, we also have some lorries to carry out this task. Therefore, the resources are the ferries, the lorries and a bridge. This bridge must be up in order to the ferries can sail under it and it must be down in order to the lorries can drive along it. Thus, the bridge is a nonshared resource *used* by both the ferries and the lorries.

The user can define the actions, resources and actions of this problem by using the specification language defined in section 2.1. For instance, in Fig. 7 appears the definition of some objects of the problem, such as the generic resource 'ferry', 'vehicle' and the actions 'sail' and 'unload. The action 'sail' transports a 'vehicle' from the margin of the river to the other one in a 'ferry' and the action 'unload' disembarks a 'vehicle' from the 'ferry'. In this example, there are not any constraints over the attributes. Even though the action durations might be disjunctive intervals, we do not use them in order to simplify the figures.

```
(defclass FERRY (subclass-of RESOURCE)
   (slot location (type PLACE))          ;where is the ferry
   (slot status (type string))           ;is it empty
   (slot load-time  (type disjunctive-interval))
   (slot sailing-time  (type disjunctive-interval))
   (slot unload-time  (type disjunctive-interval)))

(defclass VEHICLE ()
   (slot location (type PLACE))          ;where is the vehicle
   (slot on-ferry (type FERRY))          ;in what ferry
   (slot on-lorry (type LORRY)))         ;in what lorry

(defaction sail (?n-ferry ?place1 ?place2)
   (vars (FERRY ?n-ferry sailing-time ?st))
   (preconds (FERRY ?n-ferry location ?place1))
   (test (!= ?place1 ?place2))
   (effects
         (add (FERRY ?n-ferry location ?place2))
         (delete (FERRY ?n-ferry location ?place1)))
   (duration ?st)
   (resources (ferry ?n-ferry 'during ?st)
               (bridge B1 'during ?st))) ;B1 is the bridge

(defaction unload (?n-ferry ?n-vehicle ?place)
   (vars (FERRY ?n-ferry unload-time ?ut))
   (preconds (VEHICLE ?n-vehicle on-ferry ?n-ferry)
             (FERRY ?n-ferry location ?place))
   (effects
         (add (VEHICLE ?n-vehicle location ?place)
              (FERRY ?n-ferry status 'empty))
         (delete (VEHICLE ?n-vehicle on-ferry ?n-ferry)))
   (duration ?ut)
   (resources (ferry ?n-ferry 'during ?ut)))
```

Fig. 7. Fragment of the problem specification: generic resource 'ferry', 'vehicle' and actions 'sail' and 'unload'

Let us suppose that the first action the planner plans is 'unload' which uses the ferry F1. Here, the temporal network that the scheduler manages is shown in Fig. 8.

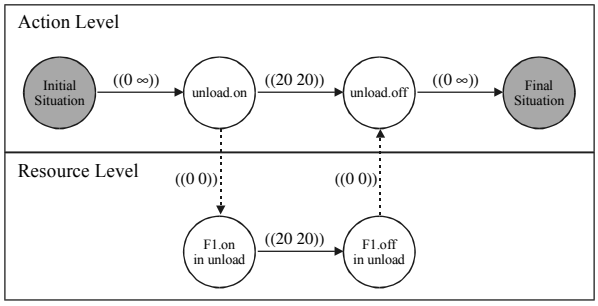

Fig. 8. Action and resource levels of the action 'unload' managed by the scheduler

Next, if the planner plans (in the same alternative plan) the action 'sail' using the same ferry F1, the new constraints asserted in the temporal network are what appear in Fig. 9. In this figure, we can see the temporal constraint *((-● -21) (16 ●))* which implies using the ferry F1 in a nonsimultaneous way. For this reason, F1 will be used in the action 'unload' either 16 units of time after its use in 'sail' or 21 units of time before, but no simultaneously.

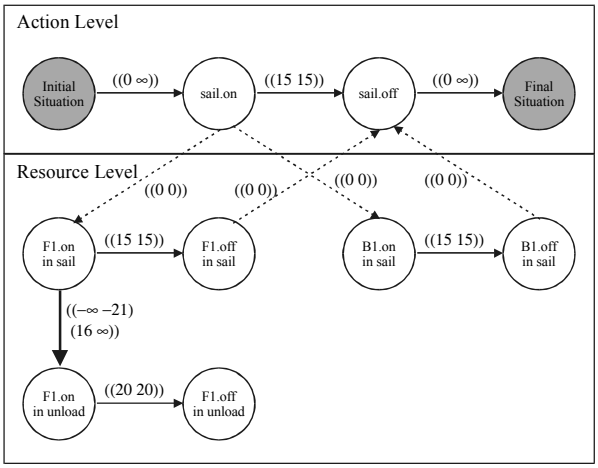

Fig. 9. Action and resource levels of the action 'sail' managed by the scheduler

3.4. Management of temporal constraints

The temporal constraints over actions, resources and attributes are treated as disjunctive metric-temporal constraints. Therefore, we have two possible alternatives to manage them and to guarantee their consistency and correctness. According to the behaviour of these constraint management algorithms we can classify them into two different kinds: algorithms that maintain the derived constraints and algorithms that only maintain the input constraints (Salido, Garrido and Barber 2000).

Algorithms that maintain the derived constraints

These algorithms require large amounts of memory to store all the generated constraints in the closure process. The reason of maintaining the derived constraints is to allow us to know quickly (without any additional process) which is the temporal constraint between two time points. When a new constraint between two temporal points is asserted into the system, these algorithms check its consistency with the existing constraint. If the new constraint does not violate the existing one, the resulting constraint will be the more restrictive combination between them. There are several levels of consistency, typically path-consistency (which guarantees the consistency of all the paths between two temporal points) and global consistency (Dechter, Meiri and Pearl 1991) which guarantees the minimality of the network.

The propagation (deriving new constraints) is carried out by means of the closure process detailed in (Barber 2000) and it is graphically represented in Fig. 10. Briefly, each time a new constraint is asserted between the time points *i* and *j*, the following loops are executed:

i) Loop 1. The derived constraint between node *i* and node k_n is calculated:

$$C_{ik_n} = C_{ik_n} \oplus (C_{ij} \otimes C_{jk_n}),$$

where the operation $\oplus$ is the intersection operation, and $\otimes$ is the combination operation (Dechter, Meiri, Pearl 1991).

ii) Loop 2. The derived constraint between node *j* and node l_m is calculated:

$$C_{jl_m} = C_{jl_m} \oplus (C_{ji} \otimes C_{il_m})$$

iii) Loop 3. The derived constraint between node l_m and node k_n is calculated:

$$C_{l_m k_n} = C_{l_m k_n} \oplus (C_{l_m j} \otimes C_{jk_n})$$

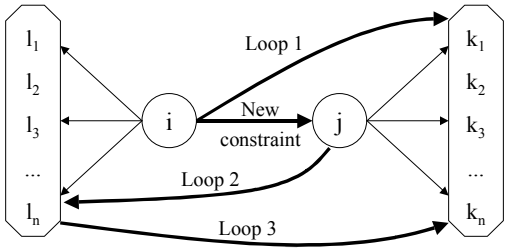

Fig. 10. Closure process which propagates the effects of asserting a new constraint

Since guaranteeing the consistency in disjunctive networks is a very complex task (NP-complete complexity), we can decrease its complexity by using algorithms (of polynomial complexity) that relax the consistency. The former algorithms perform the propagation process in a faster way, but do not guarantee a consistent solution. Hence, we cannot assure each new constraint is inconsistent.

Algorithms that only maintain the input constraints

These algorithms may be used with the aim of reducing the complexity of adding new constraints. The main advantage of these algorithms is that they do not require large amounts of memory because they do not maintain the derived constraints. These algorithms do not carry out any propagation process among the new constraint and the existing ones in the network. Therefore, they only maintain the asserted constraints. This kind of algorithms eases the process of retracting some asserted constraints into the system. When a new constraint between two temporal points is inserted into the system, the algorithm retrieves the minimal constraint between these two primitives. Next, the algorithm checks if the new constraint is consistent with the retrieved one. If it is consistent, the new constraint is accepted, and if not it is rejected. The main problem of this approach is to calculate the minimal constraint in a nonpropagated network. It is a complex task because there exists an exponential number of paths that represent the constraints to find. Currently, we are working on new algorithms to calculate this minimal constraint in a disjunctive network in an efficient way on the basis of (Salido, Garrido and Barber 2000).

4. CONCLUSION THROUGH RELATED WORK

The main drawback of temporal planners for performing scheduling tasks is that handling difficult temporal constraints over plans, actions and nonshared resources becomes in a complex task because they do not have specific temporal managers. Usually, they use a kind of *Time Point Network* (Beck and Tate 1995) to represent time constraints on time points. Although some temporal

planners can work with metric constraints, such as IxTeT (Ghallab and Laruelle 1994), traditionally they do not have the enough temporal knowledge or they do not use disjunctive constraints. In opposition, if specific processes of planning and scheduling are used, we will be able to apply characteristic features of each process. For instance, in the planner, techniques to diminish the search space and in the scheduler, more efficient criteria to carry out a better optimisation of the obtained schedule.

On the other hand, CSP and incremental CSP techniques (Tsang 1993) are not applicable enough due to the fact that they obtain a new solution instead of guaranteeing only the consistency of the problem constraints. Other authors have modelled plans as a set of constraints (future and plan entity constraints) which together limit the behaviour of the plan during its execution (Tate 1996).

As we can see, much effort has been performed in order to manage resource scheduling in an efficient way (Srivastava and Kambhampati 1999). Moreover, many attempts of integrating planning and scheduling have been carried out, mainly in the works of Muscettola and Smith with *HSTS* (Muscettola 1994, Smith 1993) and Gervasio (Gervasio and DeJong 1992). According to (Dean, Greenwald and Kaelbling 1994), we propose an integrated refinement system similar to the one proposed for (Ghallab and Laruelle 1994). Our system works with different alternative plans (*contextual scheduling*) and with disjunctive constraints to schedule resources avoiding their simultaneous usage. Hence, in this paper we have detailed the behaviour of our scheduler in our planning and scheduling integrated environment. The scheduler must have a dynamic and interactive behaviour different from traditional *CSP*s. Because the planner in a planning environment frequently provides the scheduler new constraints, the scheduler must validate them taking into account the plan they belong to. We have presented an easy way to manage all the constraints of the problem, both temporal constraints and resource usage constraints. They are managed by three very similar levels: action level, resource level and attribute level. In addition, we have discussed two ways to manage the temporal constraints: maintaining the derived constraints and maintaining only the input ones. Currently, we are working on nonpropagation techniques which allow us to retract constraints in a very efficient way. We are also studying the possibility of adding more expressive temporal constraints to the scheduler (Jonsson and Bäckström 1998).

ACKNOWLEDGEMENTS

This work is proposed in the *Intelligent Planning & Scheduling Group* of the Polytechnic University of Valencia (http://www.dsic.upv.es/users/ia/gps) and partially supported by the grant CICYT/TAP98-0345 from the Spanish government.

REFERENCES

Bäckström, C. 1998. Computational Aspects of Reordering Plans. *Journal of Artificial Intelligence Research* 9, 99-137.

Barber, F. 2000. Reasoning on complex disjunctive temporal constraints. *Journal of Artificial Intelligence Research* 12, 35-86.

Barret, A.; Christianson, D.; Friedman, M.; Golden, K.; Penberthy, S.; Sun, Y.; and Weld, D. 1996. UCPOP v4.0 user's manual, Technical Report TR 93-09-06d. Dept. of Computer Science and Engineering, University of Washington, Seattle, WA.

Beck, H. 1993. TOSCA: A novel approach to the management of job-shop scheduling constraints. *Realising CIM's Industrial Potential: Proceedings of the Ninth CIM-Europe Annual Conference*, 138-149.

Beck, H.; and Tate, A. 1995. Open Planning, Scheduling and Constraint Management Architectures. The British Telecommunication's Technical Journal, Special Issue on Resource Management.

Boddy, M. 1993. Temporal Reasoning for Planning and Scheduling. *SIGART-ACM Bulletin* 4(3), 17-20.

Currie, K.; and Tate, A. 1991. O-Plan: The open planning architecture. *Artificial Intelligence* 52(1), 49-86.

Dean, T.L.; Greenwald, L.; and Kaelbling, L.P. 1994. Time-Critical Planning and Scheduling Research at Brown University. Technical Report CS 94-41. Dept. of Computer Science, Brown University.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49, 61-95.

Dorn, J.; and Froeschl, K. eds. 1993. *Scheduling of Production Processes*. Ellis Horwood.

El-Kholy, A.; and Richards, B. 1996. Temporal and Resource Reasoning in Planning: the parcPLAN approach. *ECAI 96, 12th European Conference on Artificial Intelligence*, 614-618.

Ghallab, M.; and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In Hammond, 61-67.

Garrido, A.; Marzal, E.; Sebastiá L.; and Barber, F. 1999. Un Modelo de Integración de Planificación y Scheduling. In Proceedings of *CAEPIA'99* 1(3), 1-9.

Gervasio, M.; and DeJong, G. 1992. A Completable Approach to Integrating Planning and Scheduling. *Artificial Intelligence Planning Systems: Proceedings of the First International Conference* (AIPS 92), pp. 275-276, Morgan, Kaufmann.

Jonsson, P.; and Bäckström, C. 1998. A unifying approach to temporal constraint reasoning. *Artificial Intelligence* (102), 143-155.

Kumar, V. 1992. Algorithms for Constraint Satisfaction Problems: A Survey. *AI Magazine* 13(1), 32-44.

Laliberty, T. J.; Hildum, D. W.; Sadeh, N. M.; McA'Nutty, J.; Kjenstad, D.; and Smith, S. F. 1996. A Blackboard Architecture for Integrated Process Planning and Production Scheduling. In Proceedings of *ASME Design for Manufacturing Conference*, Irvine, CA.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. Morgan Kaufmann, San Mateo, CA.

Penberthy, S.; and Weld, D. S. 1992. UCPOP: A sound, complete, partial-order planner for ADL. In Proceedings of the 1992 International Conference on Principles of Knowledge Representation and Reasoning, 103-114. Kaufmann, Los Altos, CA.

Sadeh, N. M.; and Fox, M. S. 1996. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence* 86, 1-41.

Salido, M.A; Garrido, A.; and Barber. F. 2000. Evaluation of Algorithms to Satisfy Disjunctive Temporal Constraints in Planning and Scheduling Problems. To appear in Symposium on AI Planning and Intelligent Agents (AISB-2000).

Smith, S. 1993. Integrating Planning and Scheduling: Towards Effective Coordination in Complex, Resource-Constrained Domains. Italian Planning Workshop, Rome, Italy.

Smith, S. F.; Lassila, O.; and Becker, M. 1996. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In Advanced Planning Technology. Menlo Park, AAAI Press.

Srivastava, B.; and Kambhampati, S. 1999. Efficient Planning Through Separate Resource Scheduling. *AAAI Spring Symp. on Search Strategy under Uncertain and Incomplete Information*.

Tate, A. 1996. Representing Plans as a Set of Constraints – the <I-N-OVA> Model. In Drabble, B., ed., *Proc. Third Conference on Artificial Intelligence Planning Systems* (AIPS 96), 221-228.

Tsang, E. 1993. Foundations of constraint satisfaction. *Academic Press*.

Verfaillie, G.; de Givry, S.; and Lesaint, D. 1999. What is New in On-Line Scheduling? In *On-line scheduling characteristics (Working document)*. Available in http://www.lcr.thomson-csf.fr/projects/planet/ols-tcu.html.