Integrating Hybrid Reasoners for Planning and Scheduling

M^a Dolores R-Moreno[1], Angelo Oddi[2], Daniel Borrajo[3], Amedeo Cesta[2], Daniel Meziat[1]

[1] Departamento de Automática. Universidad de Alcalá.
Carretera Madrid-Barcelona, Km. 33,600. 28871 Alcalá de Henares (Madrid), Spain.
{mdolores, meziat}@aut.uah.es

[2] ISTC-CNR-Italian National Research Council
Viale Marx 15, I-00137 Rome, Italy.
{oddi,cesta}@ip.rm.cnr.it

[3] Departamento de Informática. Universidad Carlos III de Madrid.
Avda. de la Universidad, 30. 28911 Leganés (Madrid), Spain.
dborrajo@ia.uc3m.es

Abstract

Integrating Planning and Scheduling is becoming an increasingly interesing topic in Artificial Intelligence due to real application needs. Since, some problems allow a strict separation between planning and scheduling, the usual approach to solve the complete planning problem (a valid plan with temporal and resource information) consists on assigning time and resources (using a scheduler) to the selected and ordered activities[1] (generated by a planner). But, in other cases, time and resource assignments affect the selection and ordering of activities. Since there is usually no feedback from the scheduler to the planner about the temporal-resource invalid plans, these problems have either no solution, or become non-optimal solutions. In this paper, we present four approaches to integrate a planning system and a constraint-based system. We vary from a stand-alone planning system able to cope with some temporal-resource information towards the more integrated approach of interleaving planning and scheduling.

1 Introduction

Planning and scheduling have always been two very related tasks. Planning generates a plan (sequence or parallelization of activities) such that it achieves a set of goals and satisfies a set of domain constraints. Scheduling is an optimisation task where limited resources are allocated over time among both parallel and sequential activities such that makespan and/or resources usage are minimised.

Both fields have strong and weak points. From one hand we have the planning systems with a rich representation of the problem description (usually following the so called STRIPS representation). However, this type of representation has problems for dealing with variables with infinite values such as information about time and resources. This is so due to the fact that unbound variables in operators have to be assigned a value. Given that for each combination of values of those variables the planners generate an alternative in the search tree, those values cannot belong to an infinite range, such as a time point or resource consumption. Also, there is not an explicit language that allows to represent the basic Allen primitive relations between temporal intervals [3]. In order to avoid these problems, planners use a discrete model of time in which all actions are assumed to be instantaneous and uninterruptible. With respect to resources there is not a way to handle cumulative resources.

[1] Given the different terminology used in both fields, we will use the terms operator and activity indistinctly in this paper.

The second problem of planners for dealing with time and/or resources relates to the fact that usually when solving time-resource dependent tasks, some measure of optimality is pursued, such as makespan, and there are very few planners that can handle optimality criteria (though more and more planners are incorporating now some way of handling this).

On the other hand, scheduling systems can perfectly handle temporal reasoning and resource consumption, together with some quality criteria (usually centered around time or resource consumption) but they cannot produce the needed precedence relations among activities given that they lack an expressive language to represent the activities. Also, in case the optimality criteria is different than time-resource consumption, they cannot reason about that.

From this perspective, by combining scheduling and planning systems synergistically these weaknesses could be solved as described in [14]. The planning systems could deal with time and resources thanks to the scheduling search procedure that interleaves refinement solution (assigning values to variables) and constraint propagation (computation of implications of these assignments to others variables and elimination of inconsistent values). Also, planners can benefit of the scheduler optimization procedures. On the other direction, the planning system can provide the language and the precedence constraints for the activities to the scheduler, so that when a schedule is not valid according to time-resource constraints, the scheduler can ask the planner for another set or order of operators.

As a first step on merging both types of tasks, we have explored the possibility of using a standard non-linear domain independent planner, PRODIGY [23], and using it directly for integrating planning and scheduling. This planner does not have an explicit model of time representation nor a declarative way for specifying resource requirements or consumption. However, thanks to its capability of representing and handling continuous values variables, coding functions to obtain variables values and the use of control rules to prune the search, we have successfully integrated them in a satellite control domain [20]. This domain needs to integrate planning and scheduling for setting up nominal operations to perform in three satellites during the year. Similar approaches that allow planners to also reason about temporal and resource usage are IxTeT [15] or HSTS [18].

A second approach we have explored consists on using the O-OSCAR scheduler system [8, 11] after the QPRODIGY planner [5].[2] For a given problem, QPRODIGY generates a plan as a sequence of activities that can minimise some quality criteria. Then, the O-OSCAR scheduler obtains a viable temporal and resource solution. We provide two different solutions within this approach. In the first approach, the total-order plan that QPRODIGY generates is given directly to O-OSCAR. In the second approach, the total-order plan is converted into a partial-order plan to be given to O-OSCAR. A similar approach can be found in [21], though they did not use a planner that is able to handle by itself some types of temporal information as it is the case of QPRODIGY. Therefore, their planner is not able to avoid generating some types of temporal invalid plans.

The last approach we propose is to integrate the temporal and resource reasoning of the scheduler in an interleaved execution with the planner, by exchanging relevant information. This approach is similar to the one reported in [14]. One difference relies on the fact that our planner can handle some time-resource related information, plus can reason about some other optimisation criteria. Therefore, we can control/vary the amount of reasoning that each component (planner/scheduler) will perform.

The paper is structured as follows. Section 2 presents how the planner by itself can handle the problem. Next, the different approaches for integrating planning and scheduling are presented. Finally conclusions are drawn and future work is outlined.

2 The techniques

In this section, we will briefly describe the planner we are using, QPRODIGY, and the scheduler, O-OSCAR.

2.1 The planner. QPRODIGY

PRODIGY [23] is an integrated architecture that has been used in a wide variety of domains: from artificial domains to real applications such as satellites control. The problem solver is a non-linear planner that uses

[2] QPRODIGY is a planner based on PRODIGY that is able to handle optimisation metrics.

a backward chaining means-ends analysis search procedure with full subgoal interleaving. The planning process starts from the goals and adds operators to the plan until all the goals are satisfied. QPRODIGY is a version of the planner that is able to reason about different quality metrics, allowing to declaratively define several quality metrics in the operators, and then performing a branch-and-bound search for "good" solutions [5]. This search can be enhanced by using pre-defined or learned quality-based control knowledge. The inputs of the planner are:

1. The domain theory that contains all the actions represented by the operators. The language for describing them allows to define the usual logic formulae (including quantification), plus two useful features: variable values can be constrained using any user-defined function; and different quality metrics can be defined in each operator. With respect to the first feature, types in PRODIGY can be finite (structured in a hierarchy as in PDDL2.1 [13]) or infinite (corresponding to continuous variables, such as time, or cost). Since variables can not be given infinite possible values, values for those infinite-typed variables should constrained using functions that should return a list of possible values. To do so, they are allowed to inspect the current meta-state of the search. Therefore, one can handle continuous variables within operators, but at instantiation time, those variables will only be potentially given a finite set of alternatives.

   In Figure 1, functions are represented separatedly from the domain description. This has been done for two purposes: to emphasize the fact that this type of knowledge is important for our research purposes (as we will describe in next sections); and for uniformity with the rest of literature in planning with respect to the meaning of domain description (usually a set of operators plus a hierarchy of types).

   As an example, in the satellite domain, all data is represented in seconds since 1900 in GMT, so there is an infinite type TIME that represents this type of values and is used in the operators. The reason to use this format is for efficiency: it is faster for the planner to generate the bindings of one number variable instead of generating values for the six usual time dependent variables (corresponding to the year, month, day, hours, minutes and seconds). Also GMT is the reference zone time for the satellite company HISPASAT.

2. The second input to the planner is the problem, described in terms of an initial state and a set of goals to be achieved. In the satellite domain, the initial state describes all the events that will occur during the year such as moon blindings, sun blindings, eclipses, etc. With respect to the goals, the planner has to obtain a plan to perform all the maintenance operations along the year.

3. When there is more than one decision to be made at decision points, the third input to the planner, the control knowledge (declaratively expressed as control rules) could guide the problem solver to the right alternatives of the search tree potentially avoiding backtracking. There are three types of rules: selection, preference or rejection. One can use them to choose an operator, a binding, a goal, or deciding whether to apply an operator or continue sub-goaling.

As a result, a Total Order (TO) Plan is generated. Figure 1 shows the inputs and output of PRODIGY.

2.2 The scheduler: O-OSCAR

O-OSCAR is a scheduler that follows a Constraint Satisfaction approach [17] to solve complex multi-capacity temporal problems [8, 11]. The release used in this paper is able to solve project scheduling problems with time windows by using the ISES algorithm [10]. Its basic solving method uses a two layered problem representation:

1. The ground layer, or ground-CSP, that only represents the temporal aspects of the problem in the form of a quantitative temporal constraints network [12];

2. The higher layer is a meta-CSP, where resource conflicts are represented and reasoned about. In this layer resource constraints are super-imposed, giving rise to a set of resource conflicts that are solved with a number of sequencing decisions [9, 10].

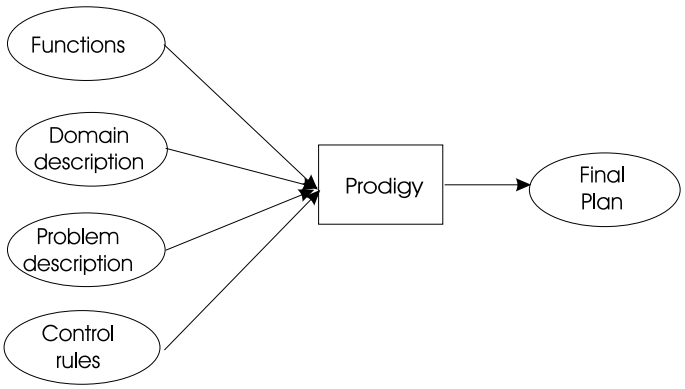

Figure 1: PRODIGY inputs and output.

The search proceeds by iteratively performing the following steps:

- Propagation of temporal consequences through the ground-CSP to compute currents bounds for all temporal variables.
- Computation of the meta-CSP identifying the set of resource capacity violations implied by the temporal network.
- Selection of a conflict in the meta-CSP according to variable ordering heuristic.
- Resolution of the conflict by imposing a new precedence constraint in the ground-CSP. This is done using a value ordering heuristic that preserves temporal flexibility.

The result is an efficient greedy procedure for generating feasible solutions to the Resource Constrained Project Scheduling Problem with time windows (RCPSP_{max}). This basic one-pass procedure is then inserted in a random restart optimization cycle that incrementally searches for better solutions by sampling new solutions on problems with reduced temporal horizons.

3 Solving problems with the planner

In the case of our study in the satellite domain, we would like to pose two types of constrains to the planner: the plan should contain details in each operator instance about time and resource assignment; some temporal-resource relations should be imposed in the solution. With respect to the first constrain, given that PRODIGY can handle continuous variables within the operators, then operators in the solutions can also incorporate some temporal information, such as start or end time of execution of operators. With respect to the second constrain, the time representation of PRODIGY is a discrete model of time, in which all actions are assumed to be instantaneous and uninterruptible. There is no provision for allowing simultaneous actions. However, the coded functions that can be used in the preconditions of operators allow to add constraints among and within operators and therefore the seven Allen primitive relations between temporal intervals can be handled [20].

As an example, how the *A meets B* Allen primitive can be imposed is shown in Figure 2. The function `add-time` calculates the end time of the operation. `DUR` is a number, integer or not, that represents the duration, and `TIME` defines the time units used (seconds, minutes, hours, days or months). `gen-from-pred` is a PRODIGY function that allows to bind the corresponding variable with all possible values that make the function's argument (a literal) match a corresponding literal in the state. Therefore, `(<dstA> (gen-from-pred (starts-at <dstA>)))` will match the literal `(starts-at <dstA>)` with the current state literals in order to assign to variable `<dstA>` all the matching values.

```
A
variables:  (<dstA> (gen-from-pred (starts-at <dstA>)))
            (<dendA> (add-time <dstA> DUR_A TIME_A))
effects:    (finished-A <dendA>)

B
variables:  (<dstB> (gen-from-pred (finished-A <dendA>)))
            (<dendB> (add-time <dstB> DURB_B TIME_B))
effects:    (started-B <dstB>)
            (finished-B <dendB>)
```

Figure 2: The *A meets B* Allen primitive.

When operator `A` is applied, it adds to the state the fact that it has finished at some point in time (`<dstA>+DUR`$_A$ in `TIME`$_A$ time units). Operator `B` binds to its variable `<dstB>` that time point, so it knows when it should start its execution.

In PRODIGY there is also no provision for specifying resource requirements or consumption. Resources can be seen as variables that can have associated values through literals that refer to them. One solution consists on restricting in the operators the set of values that can be assigned to the variable that represents the resource. For example, Figure 3 shows how the fuel that a satellite can consume in a maneuver depends on its position and the remaining fuel in the tank. The functions `calculate-fuel` and `calculate-rest-fuel` compute the fuel that the maneuver can consume and the remaining fuel after the maneuver.

```
Operator Maneuver
variables:  (<tot-fuel> (gen-from-pred (capacity-fuel <satellite> <tot-fuel>)))
            (<pos> (gen-from-pred (position <satellite> <pos>)))
            (<fuel-to-consume> (calculate-fuel <tot-fuel> <pos>))
            (<rest-fuel> (calculate-rest-fuel <tot-fuel> <fuel-to-consume>))
effects:    (capacity-fuel <satellite> <rest-fuel>)
```

Figure 3: Fuel consumed by a maneuver operation.

To represent resources with capacity contraints, we can use the scalar quantity model. The capacity constraints of a resource with uniform capacity can be calculated using coded functions as in time. To know more about the time and resource model we have used, we refer to [20].

4 Using a planner and a scheduler

The first approach that we have described in the previous section to integrate planning and scheduling within the same tool presents some problems when dealing with time and resources such as low reuse of the coded functions, that is, it is domain dependent. Also, given that values of variables can only be given in finite sets, and only at instantation of operators time, it has problems when reasoning about infinite values in intervals (functions cannot return an infinite list of variables values), and also when those values change before the actual application of operators within the planner.

In this section we present two approaches that explicitly use the scheduler, O-OSCAR, after the planner returns a plan to perform the same integration. In this case, we leave time-resource reasoning to the scheduler, or can even generate different configurations by varying the amount of time-resource management that we allow to the planner.

4.1 PRODIGY and O-OSCAR

A second approach to solve the complete planning and scheduling problem consists of integrating in sequence the planner PRODIGY and the scheduler O-OSCAR. Figure 4 shows the inputs and outputs of this approach. In this case, we remove from the domain all time-related information, letting O-OSCAR take advantage of handling the temporal information. PRODIGY generates a sequence of atemporal operations that describes the precedence relations among activities. Since PRODIGY generates a total-order (TO) plan, the precedences that PRODIGY provides to O-OSCAR are the strict order of the TO plan. Then, the plan is given to O-OSCAR that assigns a start and end time for each operator. We have built a parser to translate the PRODIGY TO plan (without temporal-resource information) into O-OSCAR inputs.

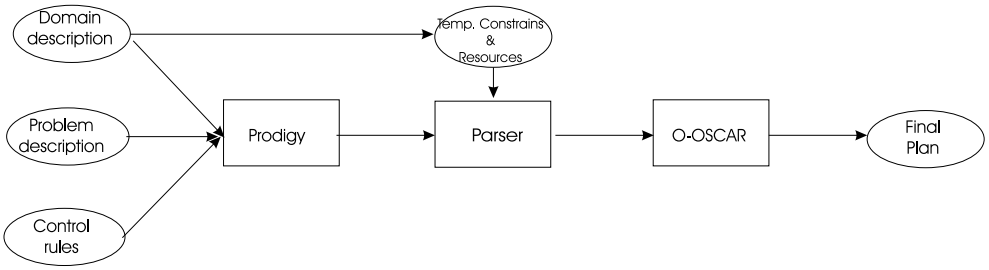

Figure 4: Sequential approach of PRODIGY and O-OSCAR.

As an example, in the satellite domain, one of the operations that is less efficiently solved by PRODIGY is performing moon blinding checks within manoeuvre operations, which requires a coded function to check moon-bloodings within two or three hours the expected time of the manoeuver. If there is any, the function should check again if twenty-four hours before and in two or three hours period could exists again a moon-blinding. Using the second approach, we can now handle it by eliminating the function that calculates the blindings from the operators descriptions, and modelling them as binary resources, i.e. sun availability, with two values: 0 if it is available and 1 when it is not available due to any type of blindings. As a result, O-OSCAR tries to locate the manoeuvre operations where the sun resource is available.

The disadvantages of this approach are:

- Not every precedence ordering between plan steps in a TO plan is necessary for maintaining its consistency, given that two activities could be executed in parallel. One way of avoiding this, would be directly using a partial-order planner, such as UCPOP [19]. However, we wanted to maintain the richer representation language of PRODIGY (that allows infinite types, for instance) together with its flexibility to easily inspect the search trees, define control knowledge, reason about quality metrics, or define new alternative control strategies (through the use of functions called handlers) [7]. We present in the next section another approach.
- The lack of communication between the two systems: in case of a failure in the scheduler, a new solution has to be generated from scratch so the computational cost is high.
- There is no way to represent explicitly in the Prodigy language, or in PDDL2.1, resources, some temporal constraints between activities or define a makespan in the solution.
- We can not take advantage of the scheduler capabilities (as minimising the makespan) since the plan obtained is very restrictive.

4.2 PRODIGY, a TO-PO algorithm, and O-OSCAR

To overcome the first drawback of the last approach, we convert the TO plan provided by PRODIGY into a Partial Order (PO) plan. A PO plan represents a set of TO plans, where each TO is a linear sequence of steps in the PO such that the PO relation in the latter is not violated by the sequence. We have followed

Procedure **Build Partial Order(TO-Plan)**
Input: TO-Plan: A totally ordered plan (being n the number of steps in the plan) and the start operator with preconditions set to the initial state.
Output: A partially ordered plan shown as a directed graph

1. **for** i=n down-to 1 **do**
   (a) **for** each *precond*∈Preconditions(op_i) **do**
   Find an operator op_j in plan that has as an effect *precond*
   Add directed edge from op_j to op_i
   (b) **for** each *del*∈Delete-Effects(op_i) **do**
   Find all operators that have as a precondition (some of the, a) delete effects of op_i
   Add directed edge from these operators to op_i
   (c) **for** each add∈Primary-Adds(op_i) (if it appears either in the goal or in the sub-goaling chain of a goal proposition) **do**
   Find all operators that delete any of the primary adds of op_i
   Add directed edge from these operators to op_i
2. Remove Transitive Edges in the graph
   Every directed edge e connecting op_i and op_j is removed if there is another path that connects the two vertices

Figure 5: Algorithm that transforms a TO into a PO.

the algorithm in [24] that takes advantage of the given total ordering of the plan, by visiting at each step, only earlier plan steps. The algorithm is sketched in Figure 5.

This greedy algorithm constructs an entirely new PO, analysing the action conditions, and using the original TO to guide the greedy strategy. It may fail to produce a minimally constrained deordering as explained in [4] because of step 1(a) in the algorithm. From all possible operators in the plan that achieve an effect needed as precondition of another operator op_j later in the plan, it always selects the closest one before op_j. Since, there may be other operators earlier in the plan having the same effect and being a better choice, the algorithm is not optimal. However, given that PRODIGY (as most current planners) is not an optimal planner either (unless all alternatives are searched for, which can be actually done in PRODIGY), then the non-optimality of the TO-PO algorithm is not a crucial point in our current state of research.

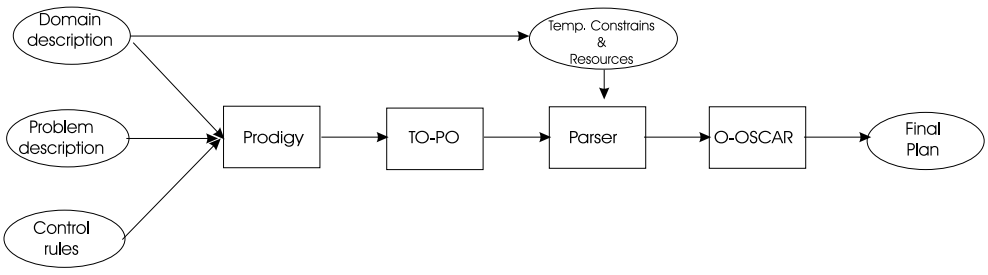

Figure 6: Sequential approach of a PO solution of PRODIGY and O-OSCAR.

This approach is shown in Figure 6. As it is also the case of the other approach, it shares the high computational cost and the lack of an explicit representation for time and resources. But there is a little

bit more communication between the planner and the scheduler and more flexibility to find an optimal solution since the plan obtained has been removed unnecessary ordering constraints. The algorithm can be extended to look for different deorderings depending on the information that the scheduler gives back. This means there are two meta level backtracking points: another deordering from the TO-PO algorithm, or another plan from the planner in case previous alternatives fail to produce a valid schedule.

5 An integrated system

The approaches studied above share the disadvantage of the lack of communication between both systems: from one part the planner provides a solution considering a given metric and the scheduler works on that solution trying to obtain an optimal assignment of time and resources. To relax the problem, we propose the hybrid reasoner shown in Figure 7. In the Figure it can be seen that both system are interleaving information during the solving process.

Reasoning is subdivided in two levels. The planner focuses on the actions selection (possibly in the optimisation of some quality metric different than time-resource usage), and the scheduler on the time and resource assignments. During the search process, everytime the planner chooses to apply an operator, it consults the scheduler for the time-resource consistency. If it is not consistent what it has previously done, then the planner will backtrack. If not, the operator gets applied, and search continues. The planner allows to choose actions under a pre-defined metric: "actions that consume less quantity of a determined resource", "actions that take less time in being executed" , "actions that maximise the capacity" etc. The scheduler also helps using other optimisation criteria during the solving process (minimising the makespan).

In recent integrations as RealPlan [22] the resource allocation is solved separately from the planning process and the quality of the plan is measured as plan length. In the approach of [14] the planner is in charge of the actions selection and the scheduler is in charge of the time and resource assignments. But one of the main features of this approach that distinguish from others relies on the degree of reasoning that can be given to each level. Since ours allows to use some type of temporal-resource usage reasoning together with some optimisation capabilities, we can control/vary the amount of reasoning that the planner will perform. For instance, we can decide to solve the temporal aspect of the problem only using the planner, and let the resource assignment to the scheduler.

Another important advantage that it is inherent to the QPRODIGY planner is the possibility to define metrics different from plan length [5], reason about those multiple criteria [1], prune the search using control rules, or automatically learn them [6, 2].

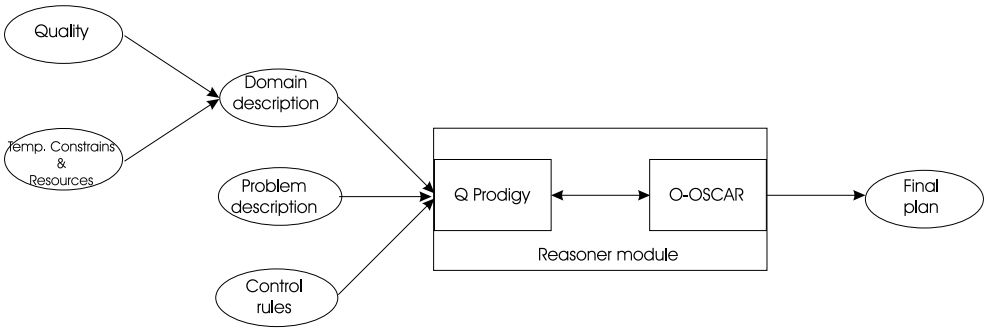

Figure 7: Structure of the integrated system.

Figure 8 shows in more detail how the different parts that are part of the reasoner module are communicated. Since Prodigy is a TO planner, a partial solution extracted from the search tree is composed of an order sequence of operators. This type of solution is not appropriate if we are looking for time and resource optimisation. To solve this, we have implemented a module that computes the causal links and threats of each operator added. The PO Solver is continuously exchanging this information with the

scheduler (formulated as a Constraint Satisfaction Problem), serialising the plan in the case of resources conflicts.

If a chosen operator gets into a non solution or inconsistent state, the planner backtracks and considers and alternative execution sequence. Backtracking to the scheduler does not mean computing from scratch the new situationm, but only eliminates the links added from the new saved state to the inconsistent state.

In this new approach the temporal-resource inconsistencies are discovered as soon as they occurs. This is due to the strong communication between the planner and the scheduler that interchange information in every point decision. Then, the computational cost decreases respect to the other approaches and the possibility to find better plans increases thanks to the capability of the CS system to allow different kind of solution analyses that can be used to guide the planning search.

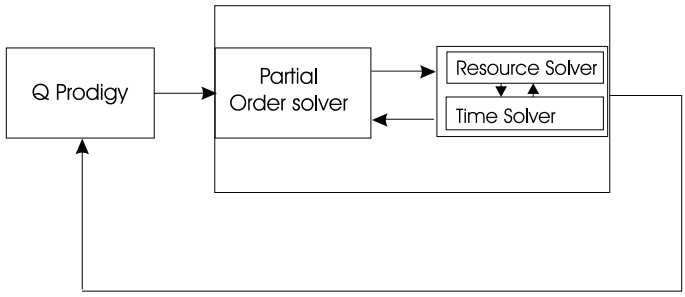

Figure 8: Components of the reasoner module.

6 Conclusions and future work

In this paper we have presented some initial steps to integrate planning and scheduling and some thoughts on possible further developments. First we have used a classical planner to reason about time and resources for a satellite domain. Then, the temporal and resource information has been eliminated from the domain, and the output generated by the planner is used as an input for a scheduler. To influence the planner choices, the scheduler should provide some information to the planner. Along this line, the last approach interleaves the refinement solution inside the search tree of QPRODIGY. In particular we are using QPRODIGY [5] in the aim of taking advantage of the quality metrics that this version is able to use. The idea is to have two systems executing in parallel; a planner and a time-resource reasoner. Information is continuously exchanged between these two representations and in particular is used to influence the PRODIGY search to prune choices that lead to either temporal or resource inconsistency. As a next step we want to compare the different approaches for several domains and compare results. We also want to use HAMLET [6] or EVOCK [2] to learn control rules for the correct decisions made during the solving problem. This has the implicit assumption that the information that is crucial (relevant) for making that decision is available in the state.

Further directions that are currently pursued concern the knowledge engineering aspects. From the experience gained in the HISPASAT project and studying the weak points of the integration, we are developing a graphical domain construction and validation tool for PDDL2.1 language as it has being done in the GIPO environment [16]. The language has been extended having in mind resources definition, temporal constraints between activities and the possibility to minimise/maximise the makespan or resources usage. The tool translates PDDL2.1 into PRODIGY's definition language. Some parts of the language are translated automatically but other parts such as the temporal functions are translated in a mixed-initiative way. Continuous attention will be devoted to an experimental comparison of the proposed approaches on the realistic benchmark represented by the HISPASAT domain.

Acknowledgements

We want to thank all the HISPASAT Engineering Team for their help and collaboration shown during the developing time of the HISPASAT project, especially Arseliano Vega, Pedro Luis Molinero and Jose Luis Novillo. The first author also wants to thank the ISTC-CNR group for their help during the visit to CNR. This work was partially funded by the CICYT project TAP1999-0535-C02-02 and a bilateral coordinated project funded by Spanish and Italian Foreign Affairs Departments. Cesta and Oddi work is partially supported by ASI (Italian Space Agency) project ARISCOM.

References

[1] Aler, R., and Borrajo, D. Learning single-criteria control knowledge for multi-criteria planning. *Proceedings of the AIPS-02 Workshop on Planning and Scheduling with Multiple Criteria* (2002), 35–40.

[2] Aler, R., Borrajo, D., and Isasi, P. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence 141* (2002), 29–56.

[3] Allen, J. Towards a general theory of action and time. *Artificial Intelligence 23* (1984), 123–154.

[4] Bäckström, C. Computational Aspects of Reordering Plans. In *Journal of Artificial Intelligence Research* (1998), vol. 9, Morgan Kaufmann Pub. Inc., pp. 99–137.

[5] Borrajo, D., Vegas, S., and Veloso, M. Quality-based learning for planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources. IJCAI Press. Seattle, WA (USA).* (2001).

[6] Borrajo, D., and Veloso, M. Lazy incremental learning of control knowledge for efficiently obtaining quality plans. In *AI Review Journal, Special Issue on Lazy Learning.* (1996), vol. 10, pp. 371–405.

[7] Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Pérez, A., Reilly, S., Veloso, M., and Wang, X. PRODIGY4.0: The manual and tutorial. In *Tech. Rep. CMU-CS-92-150, Department of Computer Science, Carnegie Mellon University* (1992).

[8] Cesta, A., Cortellessa, G., Oddi, A., Policella, N., Severoni, F., and Susi, A. Reconfigurable constraint-base architecture as a support for automating mission planning. In *ESA Workshop on On-Board Autonomy. (AG Noordwijk, The Netherlands).* (2001), pp. 219–226.

[9] Cesta, A., Oddi, A., and Smith, S. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the Fourth Int. Conf. on Artificial Intelligence planning Systems (AIPS-98)* (1998).

[10] Cesta, A., Oddi, A., and Smith, S. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics 8* (2002), 109–136.

[11] Cesta, A., Oddi, A., and Susi, A. O-Oscar: A flexible object-oriented architecture for schedule management in space applications. In *Proc. of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-99)* (1999).

[12] Dechter, R., Meiri, I., and Pearl, J. Temporal Constraint Networks. *Artificial Intelligence 49* (1991), 61–95.

[13] Fox, M., and Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *In AIPS'02 competition* (2002).

[14] Garrido, A., and Barber, F. Integrating Planning and Scheduling. In *Applied Atificial Intelligence* (2001).

[15] Ghallab, M., and Laruelle, H. Representation and Control in IxTeT, a Temporal Planner. In *Proceedings of the Second International Conference on AI Planning Systems (AIPS-94)* (1994).

[16] McCluskey, T. L., Richardson, N. E., and Simpson, R. M. An Interactive Method for Inducing Operator Descriptions. In *Proc. of AIPS'02 Workshop on Planning for Temporal Domains* (2002), pp. 151–159.

[17] Montanari, U. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences 7* (1974), 95–132.

[18] Muscettola, N. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling*, M. Zweben and M. Fox, Eds. Morgan Kaufmann, 1994.

[19] Penberthy, J. S., and Weld, D. S. UCPOP: A sound, complete, partial order planner for adl. In *In Proceedings of KR-92.* (1992), pp. 103–114.

[20] R-Moreno, M. D., Borrajo, D., and Meziat, D. An artificial intelligence planner for satellites operations. In *ESA Workshop on On-Board Autonomy. (AG Noordwijk, The Netherlands).* (2001), pp. 233–240.

[21] Srivastava, B., and Kambhampati, R. Scaling up planning by teasing out re-source scheduling. In *In Proceedings of the Fifth European Conference on Planning (Durham, United Kingdom), S. Biundo and M. Fox, Eds.* (1999).

[22] Srivastava, B., Kambhampati, R., and Do, M. B. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in RealPlan. *Artificial Intelligence 131* (2001), 73–134.

[23] Veloso, M., Carbonell, J., Pérez, A., Borrajo, D., Fink, E., and Blythe, J. Integrating planning and learning: The prodigy architecture. In *Journal of Experimental and Theoretical AI* (1995), vol. 7, pp. 81–120.

[24] Veloso, M., Pérez, A., and Carbonell, J. Nonlinear Planning with Parallel Resource Allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990), Morgan Kaufmann Pub. Inc., pp. 207–212.