Planner/scheduler interface proposal

(Rough draft – please do not circulate)

David Joslin
Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
(541) 346-0432
joslin@cirl.uoregon.edu

May 17, 1996

1 Introduction

One part of CIRL's ARPI contract is the development of a planner/scheduler interface, to be made available on the WorldWide Web. The planner/scheduler interface will allow a wide variety of planners and schedulers to interact, and should facilitate rapid prototyping, experimentation, and technology transfer.

CIRL's goal for the first year of the contract was to connect a CIRL planner and a CIRL scheduler; this has been done. The goal for the second year is to combine two non-CIRL planners and schedulers. In the third year, the interface will be made available on the Web. This document summarizes some of the challenges and issues in the design of a planner/scheduler interface, and outlines a possible solution.

2 Challenges and issues

A general-purpose planner/scheduler interface must accomodate planners that differ in many ways. A key difference is that planners and schedulers use a variety of represention. Another problem is that planners have different views on what the role of a scheduling module ought to be. This section discusses the technical challenges that arise from these differences.

Some planners (O-PLAN, SIPE) express partial plans in terms of tasks, resources, precedence relations, etc. Other planners (DESCARTES) use a representation involving assertions over temporal intervals. Low-level representations can be in terms of constraints, partially or entirely (O-PLAN, DESCARTES). Some planners maintain an agenda (O-PLAN, SIPE), which usually means that satisfying the constraints posted so far may not be sufficient to guarantee that a solution has been found, since other constraints corresponding to items on the agenda may not be taken into account. (This is what I call "passive postponement.") Other planners (DESCARTES), post constraints so that, at any point, a solution that satisfies all posted constraints will be a solution to the planning problem.

Schedulers, too, have significant representational differences. Some represent problems in terms of tasks, resources, precedence relations, etc. Other "schedulers" (interpreting the term broadly) are generic CSP solvers.

Ideally, a planner/scheduler interface would be able to accomodate all of these representations, and have a reasonable expectation of being useful for planners and schedulers that will be developed in the future. An approach that would fall short of this ideal, while still probably being of somewhat useful, would be to look for an ontology that has elements common to some subset of current planners. This was done for the "minimal ontology" defined for O-PLAN. This approach would be suitable for some planners but not for others; it would not work for either the low-level or high-level representations in DESCARTES, for example.

An alternative that would more closely approach the ideal would be a "maximal ontology" that included everything needed by all (or most) of the targetted planners and schedulers. Different planners and schedulers would use different subsets of the ontology, and translators between those views of the ontology would be required. Constructing the ontology and the translators would be a huge project, and adding a new planner or scheduler would often require that the ontology be extended, and that new translators be written.

Differences in representations are only one factor that must influence the design of a planner/scheduler interface. Architectural differences between planners, and the implications these have for the manner in which a planner most naturally interacts with a scheduling module, are another key factor. Some planners generate a scheduling problem and expect a scheduler to find a solution. The type of scheduler used here could include a general CSP solver, or a special-purpose module such as a "job shop" scheduler. In this arrangement, the planner may be making high-level decisions, such as decisions about action selection, while the scheduler makes the lower-level decisions about task ordering and resource allocation.

Other planners may only expect to use a scheduler as a consultant. Such a scheduler might perform fast, approximate tests for inconsistency, look for constraints that are common to all solutions, detect bottlenecks, apply heuristics, and so on. In this arrangement, the planner makes all or most of the planning decisions.

3 Proposed solution

This section outlines a possible solution to problem of creating a planner/scheduler interface with something close to the desired degree of flexibility. Constraint satisfaction is

```
%Domain(simplesched)
Maxtime = 3
Task(T1)
Task(T2)
Task(T3)
Resource(R1)
Requires(T1,R1)
Requires(T3,R1)
Before(T1,T2)
```

Figure 1: Example problem

the common thread that makes this possible. All of the planners and schedulers under consideration either use a CSP representation, or use a representation that can be cast as a CSP in a fairly natural way.

Rather than defining a single ontology, simple or complex, the approach proposed here uses a sorted FOL language, plus a library of "domain descriptions" expressed in that language, to make multiple, simple ontologies available. A compiler that translates a sorted FOL language into a CSP has been developed by Ari Jónsson, and will form the basis for the implementation. The availability of the compiler guarantees that, at a minimum, any planner that can represent a scheduling problem in the sorted FOL language will be able to interact with schedulers that take a CSP as their input.

In addition to the compiler, a central component of the planner/scheduler interface will be a library of domain descriptions expressed in the sorted FOL language. A domain description can be thought of as identifying the ontology that will be used for a particular interaction between a planner and scheduler.

A problem is represented in two parts: the domain name, and the problem description. The problem description, as with the domain description, is expressed using the sorted FOL language. The domain description, selected by name from the library, defines the things that are true of all problems in that domain, while the problem description defines the particular problem within that domain.

For example, suppose a planner represents partial plans in terms of tasks, resource requirements, and ordering constraints. A scheduling problem that the planner might generate for a simple problem with three tasks and one resource is shown in Figure 1.

The first line indicates that this problem uses the "simplesched" domain, and the remaining lines constitute the problem description. To keep the problem small, this example assumes that all tasks have a duration of one time unit, and resources have a capacity of one unit.

The fact that this problem is of type "simplesched" implies (as indicated in the domain description) that each task has an associated timepoint, representing the starting time for the task. The function *Time* maps tasks to timepoints. Timepoints are represented by the natural numbers from zero to *Maxtime*.

Certain axioms are also part of the "simplesched" domain description. First, the *Before* relations must induce ordering constraints on time points:

$$\forall x \forall y Before(x, y) \rightarrow (Time(x) < Time(y))$$

Second, resource conflicts must be resolved by ensuring that if two tasks require the same resource, they must not overlap.

$$\forall x \forall y \forall r (Requires(x, r) \ \wedge \ Requires(y, r)) \rightarrow (Before(x, y) \vee Before(y, x))$$

Because timepoints are represented by integers, and tasks have unit duration, tasks can only overlap if they have the same starting time. A solution to a problem of this type is an assigment of timepoints to each of the three tasks, such that the ordering and resource constraints are satisfied.

Once the planner has produced the scheduling problem shown in Figure 1, it would give that input to a scheduling module. A scheduler would have two main options for how to proceed:

Option 1: expand and solve as CSP. Given the problem description and the domain description, the sorted FOL compiler is able to generate a CSP by applying the domain description axioms to the problem description. In the example above, there would be three variables, *Time(T1)*, *Time(T2)* and *Time(T3)*, each with the domain $\{0,1,2,3\}$. The constraints generated by the axioms would be (after simplification):

$$Time(T1) < Time(T2)$$

$$Time(T1) < Time(T3) \vee Time(T3) < Time(T1)$$

The first enforces the ordering constraint *Before(T1,T2)*, and the second removes the resource conflict between tasks *T1* and *T3*.

Option 2: a specialized solver. A scheduler designed specifically to solve problems of the "simplesched" type would only need to see the input in Figure 1. For such a scheduler, the axioms in the domain description are unnecessary, since enforcing ordering constraints and removing resource conflicts is simply inherent in the algorithm. Because the problem description and the axioms are separate, this type of scheduler can solve the problem by looking only at the problem discription, ignoring the domain description.

Within either of these two options, a scheduler may try to find a solution. If a solution is found, it will be returned to the planner. If no solution is found, the returned information indicates failure; depending on the solver, this failure may indicate that the problem is unsolvable, or may only indicate that no solution was found within some search limit. The scheduler might also return additional information, such as a set of no-goods. Rather than trying to find a solution, a scheduler may instead only perform a quick, approximate test for inconsistency, returning the appropriate result. Some schedulers may only want to look at some certain part of the problem. For example, special-purpose modules might extract just the temporal ordering constraint, or just the resource constraints. A module similar to TACHYON might check for temporal consistency; another might look for resource bottlenecks.

The options available to schedulers, plus the ability to define multiple problem types and the domain descriptions associated with each, make it possible to for the planner/scheduler interface to accomodate a wide variety of representations and architectures. For a planner to make use of the planner/scheduler interface, it would need to generate problems in the sorted FOL language. At a minimum, this would allow the planner to make use of standard CSP solvers such as WSAT or NTAB. If the problem makes use of domain description in the library, it also becomes possible to submit the problem to any scheduling module that understands that type of problem.

Further flexibilty would arise from the ability to translate one type of problem to another. The "simplesched" problem type used in the example above, for example, could easily be translated into a format appropriate for a more general scheduler for which tasks can have durations longer than one unit, and there can be multiple resources of each type. This transformation would consist of modifying the problem description by adding durations of one time unit to each task, and so on, plus changing the problem type from "simplesched" to the more general case.

Making a scheduling module available requires that domain description assumed by that module, if any, be in the library. Obviously, if a scheduler requires a particular domain description, it can only be of use to a planner that can generate a problem of the appropriate type.

4 Future work

This section outlines the work that I expect to do over the next year. Because a compiler for the sorted FOL language is already available, the most difficult development task will probably be constructing a useful domain description. If a single ontology were being developed that had to serve for many planners and schedulers, and revising the ontology later would break earlier efforts, it would be important to try to make the ontology completely general from the very beginning. Because the approach proposed here allows multiple domain descriptions to be used, this burden is somewhat reduced.

If the planner/scheduler interface turns out to be useful, the number of domain descriptions may multiply. Having too many domain descriptions would then tend to diminish the usefulness of the planner/scheduler interface, because in order for a planner to be able to use a variety of schedulers it would have to accomodate a variety of domain descriptions. A different view of the situation, however, is that the planner/scheduler interface will provide an environment in which different domain descriptions compete. Hopefully, the most popular domain descriptions will turn out to be those that are actually the most useful. This "survival of the fittest" approach to refining our views on what an ontology should look like (and how many there should be), seems to me to be highly preferable over the alternative of designing an ontology by committee.

My first goal will be be to refine this proposal. Further details are obviously needed in many areas, such as the mechanism for returning information from the scheduler to the planner.

The next step will be to implement a prototype using the sorted FOL compiler. The first domain description will probably be for DESCARTES interacting with a general-purpose CSP solver.

The second domain description will be for a non-CIRL planner and a non-CIRL scheduler. This will require finding a domain description agreeable to both, perhaps along the lines of the O-PLAN minimal plan ontology. The planner is likely to be either SIPE or O-PLAN, and whichever one is the focus of this initial effort, I would like to try to define an interface that the other would find useful, also. For either, it is likely that the scheduler will have to be one that acts as a consultant, rather than one that solves entire problems.

These first two applications of the planner/scheduler interface cover a good deal of the required flexibility. The planners will make use of ontologies that differ substantially; it seems very unlikely that a common ontology could be defined. The planners will also differ in what they expect from a scheduler; DESCARTES expects the scheduler to solve an entire problem, while either SIPE or O-PLAN will most likely want to work with a scheduler that performs quick, approximate checks for consistency, or possibly other kinds of analysis such as identifying bottlenecks. If the scheduler is able to look for inconsistencies in just the temporal relations in a scheduling problem, then DESCARTES should be able to make use of this as well.

Once the planner/scheduler interface is working for these two cases, the next step (a goal for year three of our contract) will be to make it available to the ARPI community on the WorldWide Web.