Planning while Executing: a Constraint-based Approach

R. Barruffi, M. Milano, and P. Torroni

DEIS, University of Bologna, Viale Risorgimento 2, 40136 Bologna, Italy,
Tel: 0039 051 2093086 Fax: 0039 051 2093073
{rbarruffi, mmilano, ptorroni}@deis.unibo.it

Abstract. We propose a planning architecture where the planner and the executor interact with each other in order to face dynamic changes of the application domain. According to the *deferred planning* strategy proposed in [14], a plan schema is produced off-line by a generative constraint based planner and refined at execution time by retrieving up-to-date information when that available is no longer valid. In this setting, both planning and execution can be seen as search processes in the space of partial plans. We exploit the Interactive Constraint Satisfaction framework [12] which represents an extension of the Constraint Satisfaction paradigm for dealing with incomplete knowledge. Given the uncertainty of the plan execution in dynamic environments, a backup and recovery mechanism is necessary in order to allow backtracking at execution time.

1 Introduction

In dynamic and changing environments, a plan produced *off-line* by a traditional generative planner can fail during execution due to the fact that the environment can change, often in unpredictable ways. In particular, our planner works in a networked computer system environment and assembles configuration plans. The information about system services and resources cannot be complete at planning time due to its vastity and dynamicity. In these cases it is impossible to produce a complete successful plan at plan generation time and there is need to sense correct and up-to-date information at execution time in order to refine the plan. In [14], the authors propose a classification followed by a deep analysis of the main planning strategies able to integrate execution time sensory data into the planning process. The strategy we follow is called *deferred planning* consisting in delaying until execution the decisions depending on sensing. As a consequence, there is need for a sensing mechanism for testing the environment and a procedure for plan refinement at execution time. We integrate a constraint-based planner aimed at producing a plan schema with an executor able to refine the plan before executing it. Both those components are able to sense the real world by means of a constraint based framework, called Interactive Constraint Satisfaction Problem (ICSP), proposed in [12]. The main point of this paper is to describe our architecture and how it implements the *deferred planning* strategy by exploiting the ICSP framework.

2 Different Strategies to cope with Dynamicity

The enhanced complexity of traditional planning techniques when applied to dynamic environments is due to the facts that (*i*) typically the planner is not the only agent that causes changes on the system and (*ii*) often changes are not deterministic. This can lead to a failure of the plan execution, either because action preconditions are no longer verified at execution time, or because action effects are not those expected.

In [14], the authors present three different extensions to conventional planning techniques whose aim is to cope with uncertainty:

– *planning for all contingencies*, so that once sensing is performed, only the plan correspondent to the actual contingency will be executed [15, 2];
– *making assumptions*, so that planning decisions will be based only on the assumed value of the sensing result [5, 9];
– *deferring planning decisions* until information depending on sensors is available [14, 5, 8].

The appropriateness of the strategy depends on the application, and, in particular, on the criticality of mistakes, on the complexity of the domain, and on the acceptability of suspending execution to do more planning.

Our architecture follows the *deferred planning* strategy, as it will be described in the next section. The *deferred planning* approach aims at avoiding doing useless computation at planning time. Some portions of the plan requiring information which can be available only at execution time are left incomplete. In this way the planner could miss some important dependencies between the partial plans it is producing. This is why plan execution can fail and it is strongly required that the actions contained in partially specified plans are reversible.

3 An ICSP-based Planning Architecture

Our planning architecture is in charge of computing configuration plans in a networked computer system [4, 10]. The domain knowledge is composed by many different types of objects (e.g., machines, users, printers, services, files, processes), their attributes (e.g., sizes, availability, location) and relations among them (e.g., user u is logged on machine m). In this case, there is an enormous amount of knowledge to consider. In addition, this information can change during the system's life due to actions performed on the objects (e.g., removing or creating files, connecting or disconnecting machines, adding or deleting users, starting or killing processes). Thus, it is not convenient, if possible at all, to store all this information in advance and keep it up-to-date. We developed a planner able to deal with dynamic and incomplete knowledge. Our solution follows the *deferred planning* approach described in [14]. However, while in [14] the deferred decisions are represented by all the goals involving data that must be obtained through sensing, our planner does not defer until execution all the goals which require sensing since it is able to sense at planning time. In our approach deferred decisions are represented by:

– non deterministic variable bindings: since variable domain values represent alternative resources whose state can change during or after plan construction, we want to avoid as much as possible to commit to premature choices;
– acquisition of up-to-date information when that sensed at planning time is no longer valid.

Both the planner and the executor are able to sense the real system by means of a constraint based framework representing an extension of the Constraint Satisfaction paradigm and called Interactive Constraint Satisfaction Problem framework [12].

3.1 Preliminaries

Interactive Constraints (ICs) are declarative relations among variables whose domain (i.e., the set of values the variables can assume) is possibly partially or completely unknown. An interactive domain is defined as $D(X) = [List \cup Undef]$ where $List$ represents the set of known values for variable X, and $Undef$ is a domain variable itself representing (intensional) information which is not yet available for variable X. An Interactive Constraint Satisfaction Problem (ICSP) is defined on a set of variables ranging on interactive domains. Variables are linked by ICs that define (possibly partially known) combinations of values that can appear in a consistent solution. As for traditional Constraint Satisfaction Problems, a solution to an ICSP is found when all the variables are instantiated consistently with constraints. For a formal definition of the ICSP framework see [12]. ICs operational behaviour extends standard constraint propagation with a data acquisition mechanism devoted to retrieving consistent values for variable domains. In particular given a binary interactive constraint $IC(c(X,Y))$, its operational behaviour is the following:

1. **If** both variables are associated to a partially or completely unknown domain, the constraint is suspended;
2. **else, if** both variables range on a completely known domain, the constraint is propagated as in classical CSPs;
3. **else, if** one variable (say X) ranges on a fully known domain and the other (Y) is associated to a fully unknown domain a knowledge acquisition step is performed; this returns either a finite set of consistent values representing the domain of Y, or an empty set representing failure.
4. **else, if** X ranges on a fully known domain and Y is associated to a partially known one, Y domain is pruned from values non consistent with X. If Y domain becomes empty a new knowledge acquisition step is performed for Y driven by X.

This is a general framework which can be used in many applications. It is particularly suited for all the applications that process a large amount of constrained data provided by a lower level system, see for instance [11, 12].

3.2 The Algorithm

According to the *deferred planning* strategy proposed in [14], a plan schema is produced off-line by a generative planning process and refined at execution time by retrieving up-to-date information when that available is no longer valid. In this setting, both planning and execution represent search processes in the space of partial plans. More precisely the plan execution can be seen as the second phase of the same search algorithm aimed both at producing and executing a plan. The generative phase of the algorithm represents a Partial Order Planner (POP)[16] interleaving open condition[1] achievement and conflict resolution steps. As far as the open condition achievement is concerned, three alternative cases are possible: (i) the open condition is already satisfied in the initial state, (ii) it can be satisfied by an action already in the plan, (iii) there is need of a new action in order to satisfy it.

The planning problem is mapped onto an ICSP so that the planner becomes able to both exploit constraint satisfaction techniques in order to reduce the search space and deal with incomplete knowledge. The method we propose embeds knowledge acquisition activity into the constraint solving mechanism, thus simplifying the planning process in two points. First of all, there is no need to add declarative sensing actions to the plan [1, 6, 10], we provide a sensing mechanism where no further declarative action is needed apart from the causal actions. Second, only significant information for the planner is retrieved. As a consequence, variable domains are significantly smaller than in the standard case.

Open conditions are treated as ICs. Variables appearing in ICs represent system resources, and domain values represent alternative instances. Variable domains contain all the known alternative resources; they can be either (*i*) completely known, containing objects which can be assigned to the corresponding variable; (*ii*) partially known, containing some values already at disposal and a variable representing intensional future acquisitions; (*iii*) totally unknown, when no information has already been retrieved for the variable. As soon as an open condition $p(X, Y)$ is selected, the constraint solver will propagate the corresponding $IC(p(X, Y))$ to test if there exists at least one value of X and Y that already satisfies p in the initial state. When variables (X, Y) range on known domains, traditional constraint propagation is performed in order to prune inconsistent values from domain, otherwise constraint propagation results in acquisition of domain values. In order to provide Interactive Constraints with the capability to sense the system we need to associate them with appropriate information gathering procedures, working as access modules to the real world. In our environment such procedures can be represented by simple UNIX sensing commands as well as by scripts when sensory requests involve setup activities. It is worth noting that when appropriate sensors are available, Interactive Constraint retrieve only information consistent with the context so as to simplify the task of pruning inconsistent alternatives. For instance, suppose that, during the planning process, we need to locate a file *mydoc* in a UNIX system, i.e., we need

[1] An open condition is indifferently represented by a precondition or a final goal conjunct still to be satisfied.

to propagate the interactive constraint $inDirectory(mydoc, Location)$. Suppose, also, that the file *mydoc* is initially contained in three different directories $dir1$, $dir2$ and $dir3$. If variable *Location* has an unknown domain, an acquisition step is performed and those three values are retrieved (through the $find$ Unix sensing command), otherwise the domain is pruned from not consistent values (e.g $dir4$).

If a constraint fails (i.e., a variable domain becomes empty), it means that the corresponding precondition is not satisfied in the initial state (i.e., there is need of an action in order to achieve it). On the other hand, when more than one value are left in a variable domain after all possible propagation, it means that all those values satisfy that constraint in the initial state. In a traditional CS_based approach, there is need for a non deterministic labelling step in order to find a final solution. In our architecture, the labelling step takes place at plan execution time so that at the end of the generative phase variables might be associated with a domain containing more than one value.

Given the plan schema produced by the generative phase, the executor selects the first action to be executed. An interactive constraint propagation activity checks the satisfability of its preconditions in the real world. If precondition variables are already instantiated, the interaction with the underlying system results in a consistency check, while if those variables are associated to a domain, the domain can be pruned in order to remove values which are no longer consistent with the current state of the system. Value removal can trigger constraint propagation which, in turn, removes values from other variable domains, thus reducing the execution search space. If, after propagation, a domain is empty, meaning that values retrieved at planning time no longer verify the correspondent precondition p, a backtracking step is performed in order to select an alternative action or partial plan which satisfies p. When all the variables of the action range on non empty domains, necessary non deterministic labelling steps are performed and the action is executed. The same reasoning applies until all the actions are successfully executed.

3.3 An Example

Let us consider a network where a monitoring application ensures that certain processes are *up and running* (i.e. that their *status* is `on`). Once the status of the system is recognized as faulty, the planner is activated in order to provide a recovery plan.

Let us suppose that one of those processes, called `Trigger`, is `off`, and that for activating it the planner generates the plan P_1 of actions shown in Figure 1. Note that some domains are partially known, others are still completely unknown. `TriggerStart` is the daemon process in charge of activating the `Trigger` process, and its code is contained in the executable file `TMAboot`. When activating `TriggerStart`, `TMAboot` must be located in a directory (`X`) corresponding to the so-called *runlevel* (`I`) of the process. For instance, if `TriggerStart` is to be activated at runlevel `'3'`, `TMAboot` must be in a directory called `'/sbin/rl3'`. The runlevel is a parameter of the machine which is set at boot-time. In particular, in

```
***** plan to be executed: *****

killProcess(TriggerStart)
copy(TMAboot, D1, X); X:: [Undef] D1:: [/sbin/rl1, /sbin/rl2, Undef]
onTriggerStart(I, X); X:: [Undef] I:: [3, Undef]
```

Fig. 1. Plan to be executed.

order to achieve the goal of having processes `TriggerStart` and `Trigger` on, P_1 suggests that process `TriggerStart` is killed, that file `TMAboot` is copied from directory `D1` to directory `X` and that process `TriggerStart` is activated from the directory `X` corresponding to the runlevel `I`. At planning time only relevant

```
***** executing plan... *****

now checking preconditions...
        ---> condition status(TriggerStart, on) succeeded
...preconditions checked.
now doing labelling on preconditions...
labelling killProcess(TriggerStart)
...labelling on preconditions done.
now executing action 1: killProcess(TriggerStart)...
        ---> action killProcess(TriggerStart) succeeded
now checking preconditions...
        ---> condition inDirectory(TMAboot, D1) succeeded
...preconditions checked.
now doing labelling on preconditions...
labelling copy(TMAboot, D1, X)
...labelling on preconditions done.
now executing action 2: copy(TMAboot, /sbin/rl1, X)...
        ---> action copy(TMAboot, /sbin/rl1, /sbin/rl3) succeeded
now checking preconditions...
        ---> condition status(TriggerStart, off) succeeded
        ---> condition inDirectory(TMAboot, /sbin/rl3) succeeded
        ---> condition configDir(/sbin/rl3, I) succeeded
...preconditions checked.
now doing labelling on preconditions...
labelling onTriggerStart(I, /sbin/rl3)
...labelling on preconditions done.
now executing action 3: onTriggerStart(3, /sbin/rl3)...
        ---> action onTriggerStart(3, /sbin/rl3) succeeded

***** ...plan executed *****
```

Fig. 2. Output messages generated during the execution of the plan: case 1.

facts are retrieved from the world: in particular, the planner knows that the machine is `on` at runlevel 3, that four directories (`'/sbin/rl0'`, `'/sbin/rl1'`, `'/sbin/rl2'`, `'/sbin/rl3'`) exist and that they correspond to four different runlevels, that process `TriggerStart` is `on` and that process `Trigger` is `off`. Finally it knows that a copy of the file `TMAboot` is contained in two different directories (`'/sbin/rl1'`, `'/sbin/rl2'`). If the world does not change the ex-

```
now checking preconditions...
        ---> condition inDirectory(TMAboot, D1) succeeded
...preconditions checked.
now doing labelling on preconditions...
labelling copy(TMAboot, D1, X)
...labelling on preconditions done.
now executing action 2: copy(TMAboot, /sbin/rl2, X)...
        ---> action copy(TMAboot, /sbin/rl2, /sbin/rl3) succeeded
```

Fig. 3. Output messages generated during the execution of action `copy`: case 2.

ecutor will instantiate variable `D1` either to `'/sbin/rl1'` or to `'/sbin/rl2'`, variable `X` to `'/sbin/rl3'` and `I` to `3`. The output messages generated by the execution module are those of Figure 2. We can recognize different steps in the execution of each action: a first phase where the executor checks if the preconditions of the current action hold, a labelling phase where domains, if any, are labelled and eventually an execution phase, which modifies the state of the world.
If the actions are successfully performed, the world is led to a final state with all the relevant processes `on`. Now, let us suppose that before executing action `copy(TMAboot, D1, X)` some external agent in the world deletes file `TMAboot` from `'/sbin/rl1'`. The actual world contains only one instance of such file, in directory `'/sbin/rl2'`. Therefore the executor cannot label the plan in the same way as before (i.e., `copy(TMAboot, /sbin/rl1, /sbin/rl3)`). What it does, after checking in the world the domain of `D1`, is to choose one of the domain values which are actually left (i.e., `'/sbin/rl2'`). The execution proceeds as in the first case, with the only difference that the file is copied from a different source (`'/sbin/rl2'`). See Figure 3.

4 Non-Monotonic Changes

Up to now, we have considered that values acquired during plan construction can be no longer available during plan execution. However, a more complex situation occurs when some new values are available during plan execution and have not been retrieved during plan construction. Standard CSPs do not deal with value insertion in variable domains since it implies reconsidering previously deleted values which can be supported by the newly inserted value. The ICSP framework can cope with non monotonic changes of variable domains thanks to the *open domains* data structure.

An *open domain* is represented by a set of known values and a variable representing the unknown domain parts, i.e., potential future acquisitions. Thus, if, during plan execution, the entire set of known values (those acquired during plan construction), is deleted because of precondition verification, a new acquisition can start aimed at retrieving new consistent values. If no values are available, backtracking is performed in order to explore the execution of other branches in the search space of partial plans. If the overall process fails (and only in this case), a re-planning is performed.

Dynamic Constraint Satisfaction (DCS) [13] has been proposed in order to deal with non monotonic changes. DCP solvers maintain proper data structures so as to tackle modifications of the constraint store. Thanks to the ICSP framework we do not need to store additional information for restoring the constraint store consistency as done by DCS approaches. On the other hand, our method makes the propagation we perform less powerful than that performed by dynamic approaches. In fact, if we consider a constraint between variables `X` and `Y`, the variable inserted in the domain of variable `X` represents a potential support for values in the domain of variable `Y`, which cannot be pruned until the domain of `X` becomes closed.

Example. Given the example above let us consider a third case. If the domain initially retrieved for `D1` is completely wiped and an instance of file `TMAboot` is put in another directory, let us say `'/sbin/rl0'`, it is necessary to perform more acquisition via the undefined part of the `D1` domain. In particular, the constraints active on `D1` can shape from the new world setting the correct domain for it at execution time and let the plan be once more successfully executed. Figure 4 shows the output generated by the execution of action `copy`.

```
now checking preconditions...
        ---> condition inDirectory(TMAboot, /sbin/rl0) succeeded
...preconditions checked.
now doing labelling on preconditions...
labelling 2
labelling copy(TMAboot, /sbin/rl0, X)
...labelling on preconditions done.
now executing action 2: copy(TMAboot, /sbin/rl0, X)...
        ---> action copy(TMAboot, /sbin/rl0, /sbin/rl3) succeeded
```

Fig. 4. Output messages generated during the execution of action `copy`: case 3.

5 Conclusion

This paper describes an approach to *deferred planning*, which represents one of the main planning strategy to plan in presence of dynamic environments. The idea is to delay some planning decisions regarding sensory data, as much as possible, in order to reduce the gap between the world as it is observed at planning time and the world the executor performs on. We exploit the Interactive Constraint Satisfaction framework [12], which represents an extension of the CS framework based on Interactive Constraints, in order to interact with the real world. Sensing is performed both at planning and at execution time.

The implementation of this architecture has been carried out by using the finite domain library of ECL^iPS^e [3] properly extended to cope with the interactive framework. ECL^iPS^e is a Constraint Logic Programming (CLP) [7] system merging all the features and advantages of Logic Programming and Constraint Satisfaction techniques. CLP on Finite Domains, CLP(FD), can be used to represent planning problems as CSPs.

A repair mechanism is currently under development in order to cope with failures and backtracking steps over already executed actions. The repair mechanism supports all cases in which the executor realises that the effects of the action are not those expected.

6 Acknowledgments

Authors' work has been partially supported by Hewlett Packard Laboratories of Bristol-UK (Internet Business Management Department) and CNR (Project 40%).

References

1. N. Ashish, C.A. Knoblock, and A. Levy. Information gathering plans with sensing actions. In *Proceedings of the 4th European Conference on Planning*, 1997.
2. D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proceedings of AIPS-94*, 1994.
3. ECRC. *ECL^iPS^e User Manual Release 3.3*, 1992.
4. O. Etzioni, H. Levy, R. Segal, and C. Thekkath. Os agents: Using ai techniques in the operating system environment. Technical report, Univ. of Washington, 1993.
5. K. Golden. *Planning and Knowledge Representation for Softbots.* PhD thesis, University of Washington, 1997.
6. K. Golden and D. Weld. Representing sensing actions: The middle ground revisited. In *Proceedings of 5th Int. Conf. on Knowledge Representation and Reasoning*, 1996.
7. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.
8. C.A. Knoblock. Planning, executiong, sensing, and replanning for information gathering. In *Proc. 14th IJCAI*, 1995.
9. N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of AAAI-94*, 1994.
10. C.T. Kwock and D.S. Weld. Planning to gather information. Technical report, Department of Computer Science and Engineering University of Washington, 1996.
11. E. Lamma, M. Milano, R. Cucchiara, and P. Mello. An interactive constraint based system for selective attention in visual search. *Proceedings of the ISMIS'97*, 1997.
12. E. Lamma, M. Milano, P. Mello, R. Cucchiara, M. Gavanelli, and M. Piccardi. Constraint propagation and value acquisition: why we should do it interactively. *Proceedings of the IJCAI*, 1999.
13. S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, 1990.
14. D. Olawsky and M. Gini. Deferred planning and sensor use. In *Proceedings DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.
15. M. Peot and D. Smith. Conditional nonlinear planning. In J. Hendler, editor, *Proc. 1st AIPS*, pages 189–197, San Mateo, CA, 1992. Kaufmann.
16. D.S. Weld. An introduction to least commitment planning. *AI Magazine*, 15:27–61, 1994.