Propice-Plan: Toward a Unified Framework for Planning and Execution

Olivier Despouys* and François Félix Ingrand

LAAS/CNRS, 7 avenue du Colonel Roche,
F-31077 Toulouse Cedex 04, France
{despouys,felix}@laas.fr

Abstract. In this paper, we investigate the links between planning and plans execution. We propose a new approach (Propice-Plan) which integrates both activities. It implements supervision and execution capabilities, combined with different planning techniques:

– *plan synthesis* to complement existing operational plans; and
– *anticipation planning* to advise the execution for the best option to take when facing choices (by anticipating plans execution), and to forecast problems that may arise due to unforeseen situations.

This approach relies on a common language to represent plans, actions, operational procedures and constraints. In particular, the description we propose makes transitions between planning activities and execution seamless.
This work is used in two complex real-world problems: planning and control for autonomous mobile robots, and for the transition phases of a blast furnace.

1 Introduction

For years, the AI planning community has developed a wide range of techniques for *plan generation*. With few exceptions, this great amount of work did not take into account the issue of *plan execution* nor the impact execution may have on the planning process itself. Nevertheless, recent works study the links between plan synthesis and their execution. This point is indeed critical when tackling real-world problems, since several simplifying assumptions used in off-line planning reveal to be far too optimistic:

– the environment cannot be considered as *static*: unexpected events may occur during plan synthesis or its execution, thus invalidating parts of it, if not all.
– when performing planned actions, execution failures may occur; thus, the planner should take them into account to adapt its plan.

These observations motivated important research studies in order to relax some of these assumptions: probabilistic planning [Kushmeric *et al.*, 1995], possibilistic planning [Guéré and Alami, 1999], conditional planning [Pryor and Collins, 1996], Markov decision processes [Geffner and Bonet, 1998] and transformational planning [Beetz and McDermott, 1994], among others, can handle uncertainty about the environment states and possible actions outcomes. Nevertheless, these approaches still concentrate mostly on the planning process.

On the opposite side, systems and techniques dealing with plans execution have also been developed: PRS [Ingrand *et al.*, 1996], RAPS [Firby, 1994]. However, these approaches do not provide any planning activity, and fail when encountering new goals for which no explicit method is provided.

Some work has already been done to combine planning and execution. In [Wilkins *et al.*, 1994; Myers, 1998], the authors define the ACT language, a superset of the

* Part of this work is funded under contract with Usinor/Sollac, SACHEM Project.

languages used in SIPE and PRS. But the resulting system is more a concatenation of SIPE and PRS algorithms than a combined approached. 3T [Bonasso *et al.*, 1997], based on RAPS and the Adversarial Planner (AP), provides similar features in addition to a reactive skills manager. In [Levinson, 1994], the author presents Propel, which provides a unified representation for anticipation planning and execution, but here also, the operators appear to be used by either planning or execution, but not both.

Propice-Plan, the approach we propose, combines an execution model based on PRS, and various planning techniques (plan synthesis and anticipation planning) in a unified framework. In this paper, we use the term *OP* (operational plan) as the result of a planning activity, but also an operational procedure defined by a domain expert.

1.1 Application Domains

Propice-Plan is intended to be applied to perform dynamic planning and plan execution in real-world applications. Our study is motivated by an industrial collaboration for aided transition operations for a blast furnace (such as shutting it down for maintenance). These transition phases last several hours (eight hours to stop a furnace when no problem occurs). Such a process represents a dynamic continuous system for which action execution effects are poorly modelled. Nevertheless, human operators make use of a large body of operational plans which guide them through these transitional phases. They include conditional branching, loops, sub-goal posting and refinement. The set of all empirical plans represent the known paths to reach an objective. Our participation to real shutdown operations revealed that following operational plans is not sufficient and planning new sequences of actions or forecasting particular situations may help to produce better plans.

Beyond this particular application, we also use Propice-Plan on our autonomous mobile robots. Indeed, we have been using PRS for supervision and execution control [Ingrand *et al.*, 1996]. However, as pointed out by the authors in [Alami *et al.*, 1998], robotics applications also require planning activities to synthesise a particular sequence of actions to achieve certain goals. But here also, we would like to use the large body of operational plans already available for execution and control to examine and plan decision choices in advance.

1.2 Issues

Complex applications such as autonomous robots or the furnace domain require different kinds of planning techniques. In case of situations for which the expert did not specify operational plans or for which they failed, the system should try to synthesise a new one, based on the declarative information of existing operational plans (we will refer to this kind of planning as **plan synthesis**).

Besides, the execution of some operational plans may lead to critical states or may be inefficient to reach a goal in some cases. We want our system to provide some kind of look-ahead capabilities, based on these plans and their execution model, in order to advise the execution of the best option ahead, but also to opportunistically adapt plans. On contrary to XFRM [Beetz and McDermott, 1994], adaptations are not provided by a user-defined set of transformation rules, but directly derived from the model of operational plans. We will refer to this kind of planning as **anticipation planning**.

These planning activities must take into account changes in the world during the planning itself, and also produce robust plans.

Planning and execution must be integrated seamlessly. For example, from a reactivity point of view, planning should be interruptible and execution should remain the highest priority activity. From the data structure point of view, planning and execution should use a common language for plans, operators and also constraints.

The rest of this paper is organised as follows. Section 2 presents the overall organisation of Propice-Plan: the data structures and functional modules. We introduce the plan, operator and constraint representation in Section 3, followed by a brief presentation of the execution module in Section 4. Section 5 describes the planning module and Section 6 the anticipation module, which are the two planning approaches we have integrated and implemented. Section 7 concludes the paper and proposes some future work.

2 Propice-Plan Organisation

The overall architecture of our applications corresponds to the one presented in [Alami *et al.*, 1998]. This section focuses on the organisation of the decisional level and sketches the various data structures and functional components (see Fig. 1).

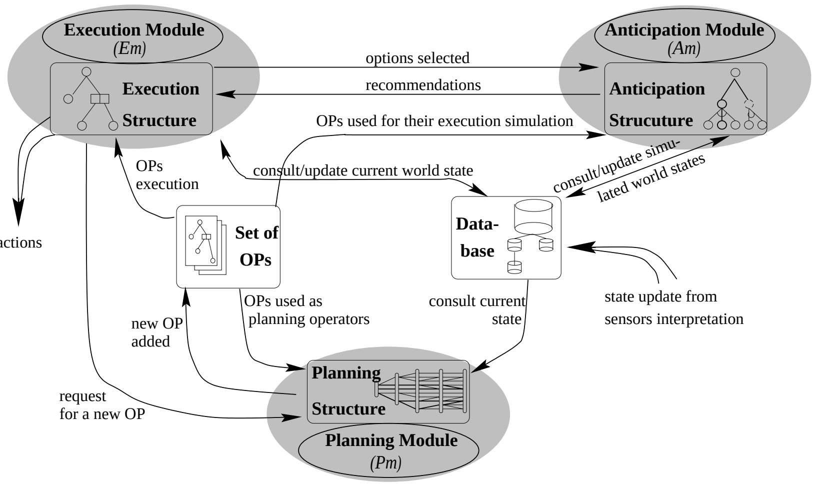


Fig. 1. Organisation of the various components.

The data structures used by the functional components follows:
• The database represents the current state of the world, as sensed by the system. It is automatically updated as new events occur. Moreover, this database provides a mechanism to handle multiple world developments, in order to enable the anticipation modules to simulate possible plan executions.
• The OPs set is initialised with the operational plans and operators which are interpreted by the execution module when receiving particular events or goals. This set can be supplemented with new OPs produced by the planning module on request.

The functional components are:
• The execution module ($\mathcal{E}m$) executes the OPs in a structure containing their execution state in response to events or to explicit goals given by the user. It is also able to recognise failures when achieving a particular goal, and then may request a new plan from the planning module.
• The anticipation module ($\mathcal{A}m$) is based on the current state of the execution structure. It simulates the execution of the OPs and evaluates the various outcomes of the different possible simulation paths to guide the $\mathcal{E}m$ if possible. These outcomes are stored and ordered according to their estimated adequacy to the current situation in the *anticipation structure* which is also consulted and updated by the $\mathcal{E}m$.
• The planning module ($\mathcal{P}m$) is currently composed of the IPP planner [Koehler *et al.*, 1997]. When asked to achieve a particular goal state, it uses the database and the set of OPs to produce a new one.

3 Plans and Operators Representation

Like Propel [Levinson, 1994], one of the goals of this work is to provide a unified representation, OPs, for planning and execution control, i.e. for operational plans and operators. We illustrate this representation, inspired to some extend by the PDDL formalism [McDermott *et al.*, 1998], with an OP (see Fig. 2) from the blast furnace application[1].

```
(defop |Purge Fluid Transfer|
   :invocation (! (in $down-sec purge-fluid))
   :call (<> (Purge-Fluid-Transfer $up-sec:section $down-sec:section))
   :context ((? (upstream-section $down-sec $up-sec))
              (! (in $up-sec purge-fluid)) (! (in $down-sec nil)))
   :effects ((=> (status $up-sec purged)) (=> (in $up-sec nil))
             (~> (in $down-sec nil)) (~> (in $up-sec purge-fluid))
             (-> (status $down-sec purged) (status $down-sec purged)))
   :properties ()
   :body ((? (upstream-valve $down-sec $up-valve))
          (? (downstream-valve $down-sec $down-valve))
          (! (valve-position $down-valve cl)) ; close downstream valve
          (if (? (drain-valve $down-sec $drain-valve))
              (! (valve-position $drain-valve cl))) ; close drain valve
          (! (valve-position $up-valve op)) ; open upstream valve
          (∧ (elapsed-time (time) @@fill-in-delay))
          (! (valve-position $up-valve cl)))) ; close upstream valve
```

Fig. 2. Example of OP.

▶ Notation
Symbols prefixed with $ denote logical variables (@@ denotes global variables). Variable typing is allowed: (syntax `<variable>:<type>`).
Modal operators may be used for goals to achieve (!), tests (?) or waiting ($\wedge$).
=>, ~> (respectively $+>$, $->$) assert or retract facts systematically (respectively under some condition indicated by the first expression).

▶ Declarative Information
The following fields represent the context of use of an OP and its known effects:
Invocation field. This corresponds to the main effect of an OP. An OP may be goal-triggered (its invocation part is prefixed with `'!'`, like in the example above) or event-triggered (without `'!'`: see Fig. 3).
Call field. This field is available to uniquely identify the OP and to bind all variables upon calling (indeed, different OPs may have the same invocation part). It allows the system to call the OP directly (thus avoiding the standard triggering/filtering mechanism).
Context field. This field gathers all preconditions required to apply the OP. As suggested in the PDDL formalism, we provide *filtering conditions* (prefixed with `?`), *feasible conditions* (prefixed with `!`) which may be made true by an other OP, and also *maintain conditions* (prefixed with `#`), which must remain true during the OP execution.
Effect field. This field contains the expected effects (apart from the one expressed in the `invocation` field) of a successful OP execution. Note that it is possible to use conditional effects.
Properties field. Some OPs may not be suitable for a particular module ($\mathcal{A}m$, $\mathcal{P}m$); this can be specified by setting a particular property in this field.

▶ Terminology
Based on the field definitions above, we define some terms used in the following sections:

[1] In this application, one has to manage a complex pipes network, in which various gas and liquids flow. This particular OP specifies how to transfer purge fluid from a pipe section ($up-sec) to the next downstream one ($down-sec).

An OP is *relevant* for a goal (or a fact) if its `invocation` part unifies with it.
An OP is *applicable* if it is *relevant* and there exists a valid unification of the `context`.
An OP is *potentially applicable* if it is *relevant* and there exists a unification satisfying all the *filtering conditions* of its `context`.
An OP is *non applicable* if it is *relevant* and there is an unsatisfied *filtering conditions* in its `context`.

▶ **Executive Information**

An OP also contains informations describing what has to be done to execute it. It is either an *action* field (linked to an external execution code), or a `body` field; the `body` is a sequence of subgoals to satisfy (test, achieve, wait) if they are not already established in the current situation, combined with conditional constructs and loops (if-then-else, while, repeat), and also parallelism (expressed with //). Execution heuristics may be specified to favour an OP among various applicable ones.

The `body` described in the example can therefore be interpreted as follows: "Determine the upstream and the downstream valves of the section $down-sec; close the downstream one; close the possibly existing drain valve; open the upstream one; wait for a given duration (corresponding to the time it usually takes to fill in a section); at last, close the upstream valve."

▶ **Discussion**

Unlike HTN representation [Nau *et al.*, 1999; Currie and Tate, 1991], OPs emphasise the dynamic and execution aspects of operational plans. For example one can explicitly write an OP which loops over a sequence of subgoals until a condition is true, or which follows one execution path or another according to a condition.

The OP description is limited to its abstraction level, regardless of lower level information; for example, the OP above does not clarify the valves positions in the `context` or `effects` fields. In this sense, the OP representation is *incomplete*, and corresponds to the *relaxed models* [Yang, 1997] used in hierarchical planning. Even if a sequence of OPs seems correct for some level of abstraction, interaction problems between effects and expected preconditions may arise at lower ones. Although it may be possible for each OP to compile off-line its extended `effects` and `context` fields, with respect to OPs used in its `body`, this technique would lead to a prohibiting number of highly specific OPs. Another solution is developed in § 6.

At last, as shown on Fig. 3, the representation proposed here makes it possible to define domain-specific constraints (ex: for security).

```
(defconst |Upper Bound For Surface Temperature|
   :invocation (surface-temperature $bf:furnace $temp:temperature-threshold)
   :context ((? (greater $temp @@max-surf-temp)))
   :body (// (! (coal-load-policy $bf suspend))
             (! (vapor-injection $bf (* @@u-vap (val-temp $temp))))
```

Fig. 3. Example of constraint.

4 Supervision and Execution Control

The $\mathcal{E}m$ uses all the data structures presented in Fig. 1. Its algorithms are heavily inspired from the PRS and we invite the reader to check [Ingrand *et al.*, 1992; Ingrand *et al.*, 1996] for a more detailed account of how it works. Roughly, its main loop runs as follows: it takes into account new events in the database and new goals. It checks sleeping and maintained conditions in the execution structure, selects appropriate OPs and constraints according to these new events or goals, and the recommendations given by the anticipation structure. Then, it places the selected OPs in its execution structure to execute them (step by step to make the execution loop tight). It results in a primitive action, the assertion of new facts in the database, or the establishment of a new goal.

The $\mathcal{E}m$ differs from PRS in various ways. It explicitly takes into account the recommendations made by the $\mathcal{A}m$ in the anticipation structure (either OP , unification choice) and updates the anticipation structure accordingly. Moreover, it may request a new OP from the $\mathcal{P}m$ when it has to achieve a goal or a conjunction such that no corresponding OP is successfully applicable.

5 Plan synthesis

As mentioned in Section 1, one of our goal is to use and integrate, when possible, existing efficient planning paradigms. We choose one of Graphplan [Blum and Furst, 1997] successors: IPP [Koehler *et al.*, 1997], which provides some interesting extensions for real world applications.

5.1 Principle

When Propice-Plan has to face a situation for which there is no applicable OP, it may request help from the $\mathcal{P}m$. This planner takes as inputs the current contents of the database (initial state), the desired situation (goal), and the OPs available for planning.

For each OP, the $\mathcal{P}m$ only uses the *declarative informations* (see Section 3). If no solution is found, it reports a failure. Otherwise, a plan is sent as a new OP (in the OPs set) with a `body` corresponding to the sequence of OPs to execute (referenced by their `call` fields), and with its `context` set to the conjunction of the facts which entail the plan (such conjunction is computed during the final backchaining process in IPP). Then, the $\mathcal{E}m$ will check the returned plan `context` (i.e. its applicability) before executing it, since the environment state may have evolved since the plan was produced. Fig. 4 shows an OP synthesised by the $\mathcal{P}m$.

```
(defop |IPP air-regulation_bf1_thres-20000 (0)|
    :invocation (! (air-regulation bf1 thres-20000))
    :context ((? (status reg12 ok)) (? (air-feeder reg12))
                  . . . (? (bf-regulation reg12 bf1)))
    :body ((// (<> (Downstream-Section-Connection c12 reg12 tc1))
                   (<> (Upstream-Section-Connection c11 reg11 tc1)))
               (<> (Regulation-Feeder reg12 reg11 sw1))
               . . .
               (<> (Air-Regulation bf1 reg12 thres-20000)))
    :properties ((op-type synthesised)))
```

Fig. 4. Example of a plan returned by the $\mathcal{P}m$.

5.2 Selecting Relevant Information

Planners based on Graphplan algorithm suffer from a drastic drop in performance when the number of operators and the size of the initial state increase [Nebel *et al.*, 1997]. Contrary to most "toy" examples where planning problems are well-conditioned, our applications have a wide range of OPs available for planning and a large number of facts used to describe the current state. As a consequence, we need an effective selection method to preserve IPP efficiency.

Our first attempt consisted in using the RIFO technique in IPP [Nebel *et al.*, 1997] to determine all operators and initial facts that seem relevant to the goal. Although this is very effective in most cases, it does not preserve the solution existence for some problems (which we encountered); this leads us to consider another method: off-line preprocessing is used to cluster the set of OPs according to their `effects` and `context` fields. As OPs are loaded in Propice-Plan, various clusters can be determined by analysing the attributes and the types of the variables in those fields. For the sake of concision, this technique will not be developed in this paper.

5.3 Monitoring the Initial State

The environment being dynamic, the $\mathcal{P}m$ has to monitor changes occurring in the database which may invalidate the initial state on which the planning graph is being built. This state is in fact restricted to those attributes described in the declarative informations of the selected OPs. We have been studying a technique to adapt the $\mathcal{P}m$ planning graph *on the fly*; the underlying idea is to stop the planning process in a stable state (ie. in the latest proposition level developed so far, once it has been complemented with mutual exclusive relations among its node, see [Blum and Furst, 1997] for more details) and clean off the rest of the graph. Then, starting from the initial proposition level, we remove successively all outdated facts, with the mutex relations they were possibly linked to, all operators they were precondition of (including noop), and restart the process for the next proposition level to remove all effects of deleted operators. This will stop when an empty level or the very last propositional one has been reached. The next step consists in completing the first proposition level with the missing facts described in the initial state, and restart IPP's forward-chaining process to complete its successors until the goal is found. At last, the plan search in the graph is performed as in IPP original algorithm. Unfortunately, this method does not guarantee the plan synthesis ending. In such a situation, we claim that the knowledge elements used for planning are inadequate with respect to the application environment dynamic.

5.4 Constraints Management

As illustrated on Fig. 3, Propice-Plan provides means to express the constraints of the domain. Thus, the $\mathcal{P}m$ provides basic constraint management mechanisms. Constraints (for instance `(surface-temperature $temp)` with $temp higher than @@max-surf-temp) are checked successively for each proposition level, when building the planning graph. If one is violated, the undesired facts are discarded in the current propositional level by use of IPP's Mutex relations (inherited from Graphplan's).

Of course, this technique is inadequate for constraints between more than two facts, and only unary and binary constraints are taken into account in Propice-Plan. However, this did not reveal too restrictive for the applicative domains we encoded so far. Moreover, this approach could be complemented with recent works using dynamic CSP techniques during the plan extraction phase [Kambhampati, 1999].

6 Anticipation Planning

The $\mathcal{A}m$ uses the OPs set and the database to simulate and examine in advance a number of possible options available to the system ahead of the execution.
It provides two types of information useful to the $\mathcal{E}m$:

- it can evaluate in advance choices in order to advise the $\mathcal{E}m$ for the best option with respect to the current state of the system, and the set of projections developed so far (cf. 6.1).
- it can anticipate some unsatisfied preconditions to come, and try to establish them with an adequate *opportunistic* strategy (cf. 6.2).

Supervision and execution systems such as those mentioned in Section 1 do not perform explicit planning. They merely choose among possible available options at run-time (such as different applicable plans for a specific goal) without any projection of the current situation in the future. Our goal is to take advantage of the large number of OPs available and of the spare time left while performing control execution and supervision.[2]

[2] For example, a blast furnace shutdown lasts eight hours and the supervision/execution process is idle most of the time. However, when a decision has to be taken, the response time must not exceed one minute.

The whole idea of anticipation planning is thus based on the simulation of OPs executive information. However, one has to keep in mind that some actions, mostly those related to sensing the value of an attribute, are non deterministic. As a consequence, we need to treat these attributes accordingly.

6.1 Guiding the $\mathcal{E}m$ Through Choice Points

Choices and Preferences For a given goal, the $\mathcal{E}m$ uses execution heuristics to choose among various applicable and relevant OPs for the most trustworthy one; though suited for most situations, the selected operational plan is generally more expensive than others (in terms of execution duration, resources,...). The first role of the $\mathcal{A}m$ is thus to evaluate such choice points before the $\mathcal{E}m$, with anticipation heuristics indicating less reliable but possibly cost-effective OPs. The corresponding results are gathered and regularly updated in the anticipation structure which contains all the projections examined so far, which will be consulted later by the $\mathcal{E}m$ when it faces a choice point. If the anticipation structure contains an option that led to a successful execution simulation, the $\mathcal{E}m$ uses it. Otherwise, the $\mathcal{E}m$ uses execution heuristics to select an adequate OP.

A specific processing is required when loops and conditional branching are simulated. Indeed, the condition value may be unpredictable due to non determinism (if (? (& (nitrogen-flow $nf) (< $nf 12))) ...). Then, there is no mean to foresee which execution branch will be performed by the $\mathcal{E}m$. Both branches are then simulated and labeled with the corresponding constraints (here: one with a nitrogen flow value less than 12, and one for the opposite).

$\mathcal{E}m$ and $\mathcal{A}m$ Interactions The initial state of the anticipation process is based on the current environment state. Then, when evaluating all possible outcomes for a choice point, the $\mathcal{A}m$ will develop different states accordingly; these will in turn lead to different options, when the next choice point will be processed. Therefore, the projections examined by the $\mathcal{A}m$ is structured as a *projection tree* (see Fig. 5). Then, assigning instantiated OPs to choice points is equivalent to searching a path in this tree, and search control is performed by anticipation heuristics.

In return, the $\mathcal{E}m$ synchronises the anticipation process with the actual environment state by updating the database (and therefore the projection tree root). Obsolete paths are automatically cut off by the anticipation structure; indeed, edges corresponding to a choice option are systematically labeled with the condition enabling this option (for example, when various OP instances are relevant for a goal, each outgoing edge from the current choice point is labeled with the corresponding `context` fields; see Fig. 5). Updates are processed every time the $\mathcal{E}m$ steps across a choice point even if it followed the $\mathcal{A}m$'s advise to take unexpected events into account.

6.2 Opportunistic Preconditions Achievement

The operational plans in the OPs set cover a wide range of expected situations including emergency ones (see Fig. 3). Yet, if the $\mathcal{E}m$ keeps executing and refining OPs without any anticipation, some preconditions required at a lower execution level may lead to inefficient plan, as they were unforeseen at the higher level.

• *Detecting inadequacy.* The $\mathcal{A}m$ detects an OP inadequacy if its execution simulation failed, eg. one of the goal in its `body` field cannot be reached. If it occurs that the relevant OPs for this goal are only potentially applicable (as defined in Section 3), the $\mathcal{A}m$ will try to adapt the current plan.

• *Fitting the current plan.* This is done by inserting, at the best place in the current plan, the proper OP which establishes the missing precondition(s). This adaptation should occur according to an *opportunistic* and *conservative* strategy, to modify the original plan as little as possible and minimise harmful interactions due to these modifications. The following example will illustrate these ideas.

Example This example was inspired by a real problem that occurred during a furnace shutdown, while purging a pipes network. A known defective section valve could not be closed in time because the operational plan dealing with such valves requires an operator with a special tool. Without anticipation, such precondition could not be foreseen and an inefficient plan execution resulted (indeed, the shutdown was delayed by an half hour to send an operator on site).

▶ **OPs and Initial State**

To illustrate this, we introduce new OPs which `body` (not figured) will not be simulated for the sake of concision. Two OPs set valves positions: either manually (for defective valves that require assistance from an operator equipped with a toolbox), or automatically from a console, if the valve is not defective. The two others manage the operator equipment and location.

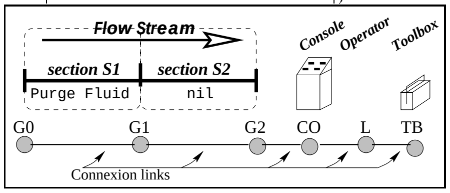
```
(defop |Automatic Valve Control|
 :invocation ((! (valve-position $v $p)))
 :call (<> (Automatic-Valve-Ctl $v:valve
                    $p:pos $old-p:pos))
 :context ((? (status $v OK))
           (? (valve-position $v $old-p))
           (! (ope-loc CO))) ;at the console
 :effects ((~> (valve-position $v $old-p))))
```

```
(defop |Get Toolbox|
 :invocation ((! (has-toolbox)))
 :call (<> (Get-Toolbox $l:location))
 :context ((? (toolbox-loc $l))
           (! (ope-loc $l)))
 :effects ())
```

```
(defop |Manual Valve Control|
 :invocation ((! (valve-position $v $p)))
 :call (<> (Manual-Valve-Ctl $v:valve
                    $p:pos $old-p:pos))
 :context ((? (status $v defective))
           (? (valve-position $v $old-p))
           (! (has-toolbox))
           (! (ope-loc $v)))
 :effects ((~> (valve-position $v $old-p))))
```

```
(defop |Go To|
 :invocation ((! (ope-loc $dst)))
 :call (<> (Go-To $dst:location $l:location))
 :context ((? (connex $l $dst))
           (! (ope-loc $l)))
 :effects ((~> (ope-loc $l))
       (+> (has-toolbox) (toolbox-loc $dst))
       (−> (has-toolbox) (toolbox-loc $l))))
```

A heuristic function is defined to minimise potentially dangerous human interventions (prefer |`Automatic Valve Control`| to |`Manual Valve Control`|). Let us consider a simple domain with two sections (S1, S2) and three valves (G0, G1, G2); the topology is described in the figure (so the related facts are not enumerated). To simplify, a valve refers both to an object and the corresponding location. The initial state ($\mathcal{I}$) is defined as follows: `(valve-position G`i` cl)`$_{i\in\{0,1\}}$ `(valve-position G2 op)` `(toolbox-loc TB)` `(status G`i` OK)`$_{i\in\{0,2\}}$ `(status G1 defective)` `(in S1 purge-fluid)` `(in S2 nil)` `(ope-loc L)`.



▶ **OP Analysis**

When loading OPs in Propice-Plan, off-line processing makes it possible to determine OP hierarchies and cluster them according to their abstraction level. This is then used to guide plan adaptation as illustrated in this example. The following table presents the corresponding two-level OP hierarchy, with related facts.

Higher level	Purge-Fluid-Transfer	*in, status:section*
Lower level	Manual-Valve-Ctl,Get-Toolbox, Automatic-Valve-Ctl,Go-To	*valve-position,ope-loc, status:valve,has-toolbox*

▶ **Projections**

Suppose we simulate the execution of an OP such that the next goals are `(! (ope-loc CO))` (to control valves positions) and then `(! (in S1 nil))` (with the |Minimise Human Risks| heuristic), and the initial state is ($\mathcal{I}$) described above. The projection tree is represented in Fig. 5.

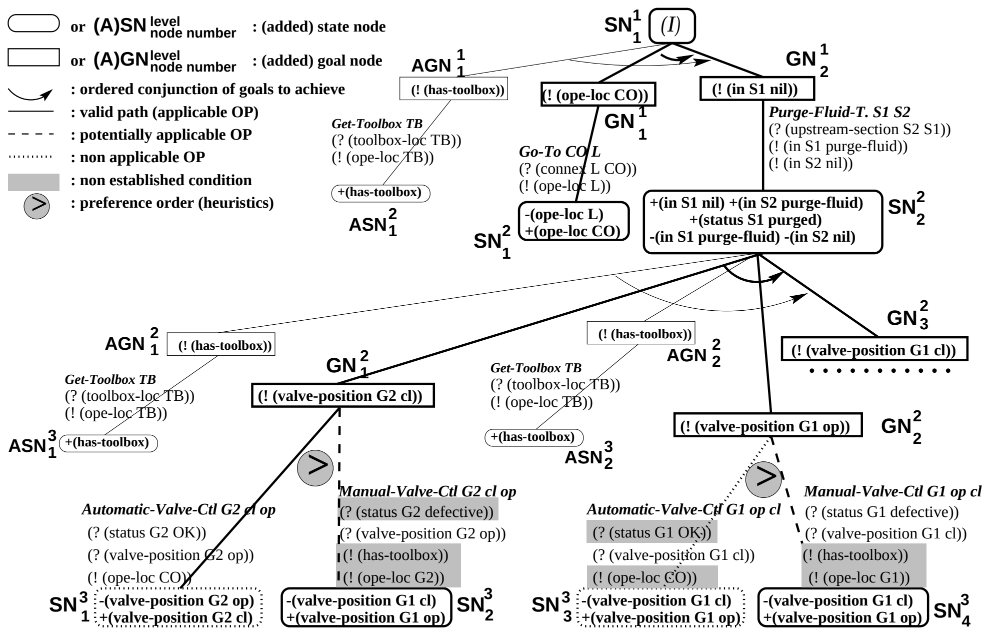

or (A)SN level node number : (added) state node
or (A)GN level node number : (added) goal node
: ordered conjunction of goals to achieve
: valid path (applicable OP)
: potentially applicable OP
: non applicable OP
: non established condition
: preference order (heuristics)
SN 1 1 (I)
AGN 1 1 (! (has-toolbox))
Get-Toolbox TB (? (toolbox-loc TB)) (! (ope-loc TB))
+(has-toolbox) ASN 2 1
(! (ope-loc CO)) GN 1 1
Go-To CO L (? (connex L CO)) (! (ope-loc L))
-(ope-loc L) +(ope-loc CO) SN 2 1
(! (in S1 nil)) GN 1 2
Purge-Fluid-T. S1 S2 (? (upstream-section S2 S1)) (! (in S1 purge-fluid)) (! (in S2 nil))
+(in S1 nil) +(in S2 purge-fluid) +(status S1 purged) -(in S1 purge-fluid) -(in S2 nil) SN 2 2
AGN 2 1 (! (has-toolbox))
Get-Toolbox TB (? (toolbox-loc TB)) (! (ope-loc TB))
ASN 3 1 +(has-toolbox)
GN 2 1 (! (valve-position G2 cl))
(! (has-toolbox)) AGN 2 2
Get-Toolbox TB (? (toolbox-loc TB)) (! (ope-loc TB))
+(has-toolbox) ASN 3 2
GN 2 3 (! (valve-position G1 cl))
(! (valve-position G1 op)) GN 2 2
Automatic-Valve-Ctl G2 cl op (? (status G2 OK)) (? (valve-position G2 op)) (! (ope-loc CO))
Manual-Valve-Ctl G2 cl op (? (status G2 defective)) (? (valve-position G2 op)) (! (has-toolbox)) (! (ope-loc G2))
Automatic-Valve-Ctl G1 op cl (? (status G1 OK)) (? (valve-position G1 cl)) (! (ope-loc CO))
Manual-Valve-Ctl G1 op cl (? (status G1 defective)) (? (valve-position G1 cl)) (! (has-toolbox)) (! (ope-loc G1))
SN 3 1 -(valve-position G2 op) +(valve-position G2 cl)
-(valve-position G1 cl) +(valve-position G1 op) SN 3 2
SN 3 3 -(valve-position G1 cl) +(valve-position G1 op)
-(valve-position G1 cl) +(valve-position G1 op) SN 3 4

Fig. 5. Example of a Projection Tree.

Projection trees are layered, with two kinds of nodes appearing alternatively: *state nodes* represent the successive process states and *goal nodes*. Initially, the tree root SN_1^1 is a state node representing the database contents, and its children correspond to the sequence of `invocations` to be performed (here, there are therefore two goal nodes GN_1^1 and GN_2^1 to simulate `(! (ope-loc CO))` and `(! (in S1 nil))`). Projections are developed as follows: various OPs may be relevant for a non established goal GN_i^l, leading to new state nodes SN_j^{l+1}; the corresponding edges $\overline{GN_i^l SN_j^{l+1}}$ are labeled with the instantiated `call` and `context` fields, and ordered according to heuristics. A new state node SN_j^{l+1} contents is only *the difference with its parent state node* at level l; `'+'` (resp. `'-'`) signs indicate added (resp. deleted) effects, according to the *declarative informations* of the OP from edge $\overline{GN_i^l SN_j^{l+1}}$. Then, goal nodes at level $l+1$ (GN_k^{l+1} nodes) are obtained by developing the `body` field of the OP from edge $\overline{GN_i^l SN_j^{l+1}}$. This process repeats until no state node could be further developed (terminal node); this occurs when the `body` field of the OP leading to this node is empty or the OP is only simulated through its `effects` field (OP marked as not relevant for anticipation). Finally, a state node is *completed* if it is terminal, or all its children are *completed* (AND-node semantic); a goal node is *completed* if at least one child is *completed* (OR-node related to choice points). The simulation is considered successful if the root node is completed; then, the $\mathcal{E}m$ can be advised for all these choice points.

The result of the simulation process corresponds to **thick edges and nodes** in the projection tree. `Go-To` and `Purge-Fluid-Transfer` are the only relevant OPs for `(! (ope-loc CO))` and `(! (in S1 nil))`, and they are applicable. For the second one, the `body` indicates that the goals `(! (valve-position G2 cl))`, `(! (valve-position G1 op))`, and `(! (valve-position G1 cl))` must be achieved. Two OPs are relevant for each of the first two ones, and the preferred one is `Automatic-Valve-Ctl`, which is applicable for the goal `(! (valve-position G2 cl))` (goal node GN_1^2). Concerning the goal `(! (valve-position G1 op))` (GN_2^2), the first relevant OP is non applicable, and the second (`Manual-Valve-Ctl`) is only potentially applicable. Thus, adaptation is necessary to establish the required conditions: `(has-toolbox)` and `(ope-loc G1)`.

▶ **Adaptation**
The basic idea is to select a flaw (unsupported feasible condition), look for all possible fixing OPs, and add the most adequate one where it suits the best (fix selection and positioning are performed at the same time, using a cost criterion). The main step of the adaptation algorithm are:

Step 1. Let $\overline{GN_i^lSN_j^{l+1}}$ be the first potentially applicable OP appearing while developing the projection tree, and such that it is not marked as non recoverable, and the tree can still be repaired (initially, all flaws are considered recoverable).
If no such OP exists, either there is a valid path in the tree (all inconsistencies are removed: exit successfully), or it is impossible to fix the flaw (report failure).

Step 2. Let $\mathcal{S}$ be the set of goal nodes AGN_m^n that may fix the flaw $\overline{GN_i^lSN_j^{l+1}}$, and added with respect to hierarchy considerations. Set a causal link between every element AGN_m^n in $\mathcal{S}$ and $\overline{GN_i^lSN_j^{l+1}}$. For every couple of elements in $\mathcal{S}$, set a mutual exclusion link.

Step 3. For each element AGN_m^n in $\mathcal{S}$, develop all possible edges $\overline{AGN_m^nASN_p^{n+1}}$ OPs corresponding to OPs relevant for AGN_m^n. Associate to those edges their insertion cost. Let $\overline{AGN_m^nASN_p^{n+1}}$ be the minimal relevant OP with regards to the associated costs.

Step 4. If the insertion cost for $\overline{AGN_m^nASN_p^{n+1}}$ is infinite, then determine the flaw $\overline{GN_i^lSN_j^{l+1}}$ that led to AGN_m^n addition (use causal links), mark $\overline{GN_i^lSN_j^{l+1}}$ as non recoverable, remove all goal nodes added because of $\overline{GN_i^lSN_j^{l+1}}$ (use causal links), and go back to step 1. Otherwise, remove all goal nodes linked by a mutual exclusion relation to AGN_m^n, and exit to restart the anticipation process from node AGN_m^n.

The first selected flaw is $\overline{GN_2^2SN_4^3}$ (Step 1), and the condition to establish is `(! (has-toolbox))`. Then, the set $\mathcal{S}$ of added goal nodes (Step 2) is $\{AGN_1^1, AGN_1^2, AGN_2^2\}$ (thin edges and nodes on the Fig. 5); no such goal node can be immediately before GN_2^1, because they do not belong to the same hierarchic level (`in` is higher than `has-toolbox`). Causal links between a flaw and its fixes are used to suppress all added nodes if it appears that the flaw cannot be recovered; a mutual exclusion link indicates that as soon as a correct fix is found, the others are useless. Edges developed in step 3 are $\overline{AGN_1^1ASN_1^2}$, $\overline{AGN_1^2ASN_1^3}$ and $\overline{AGN_2^2ASN_2^3}$. A cost is associated, estimating the OP applicability, and the interactions it may have with the rest of the plan:

- **Applicability factor.** If the added OP is applicable, this factor is zero; if it is non-applicable, it is set to ∞; otherwise, it is an estimate of the number of OPs needed to establish invalid conditions. Instead of solving this planning problem, we focus on a relaxed one where delete-lists are not taken into account, and conditional effects are turned into systematic ones; the resulting plan may not be correct, but it provides a rough idea of an added OP adequacy to the current situation.
- **Interaction factor.** If the effects of the inserted OP invalidate a condition required to execute another one (this is determined by using condition labels associated to edges), this factor is set to the estimate number of OP needed to re-establish it (same technique as above); otherwise, it is zero.

In this example, the global cost for the fixes is:

	Applicability	**Interaction**
$\overline{AGN_1^1ASN_1^2}$	2 OPs: `(Go-To TB L)` + `(Get-Toolbox TB)`	no OP
$\overline{AGN_1^2ASN_1^3}$	3 OPs: `(Go-To L CO)` + `(Go-To TB L)` + `(Get-Toolbox TB)`	no OP
$\overline{AGN_1^2ASN_1^3}$	3 OPs: `(Go-To L CO)` + `(Go-To TB L)` + `(Get-Toolbox TB)`	no OP

$\overline{AGN_1^1ASN_1^2}$ is therefore selected, and the two others fixes removed (step 4). The simulation process is started again to update the projection tree (indeed, new conditional effects may have been triggered); if simulation is unsuccessful, the adaptation process is run again. For example, the next flaw will thus be $\overline{AGN_1^1 ASN_1^2}$ (to establish `(! (ope-loc TB)))`. The final plan is therefore to have the operator go and get the toolbox in TB, go to CO via L to control the valve G2, and finally go to G1 (via G2) to manœuvre the defective valve.

Without anticipation planning, the $\mathcal{E}m$ would have closed G2 and failed to open G1; it would have then requested help from the $\mathcal{P}m$. However, the additional actions from the returned plan would have been more numerous than the ones added by the $\mathcal{A}m$, since the planning phase is launched *after an actual execution failure* (the operator being at CO, he would have had more moves to do to get the toolbox and go back to G1). On the opposite, anticipation provides rather light adaptations, which are integrated seamlessly in the original procedure.

7 Conclusion

The proposed approach addresses general issues about planning and execution in real-world domains. Our work is meant to be generic and uses two different case studies (the blast furnace and the autonomous robots domains) which showed that one single planning technique is not sufficient to cope with the wide variety of problems occurring in these domains. Therefore, Propice-Plan combines existing approaches (IPP for planning and Propice — an extended version of C-PRS — for execution) with new ones dealing with anticipation planning.

This paper presents several improvements for existing techniques to meet real-world requirements. In particular: an efficient information filtering mechanism to select only relevant facts and operators for a given planning problem; a cooperative approach between the execution and planning modules to handle possible changes concerning the initial planning state; an anticipation of OPs execution to foresee problems to come and make choices in advance; an advisable execution module that takes external recommendations into account when facing choices.

The OP formalism we adopted encompasses the notions of operational plans, operators and even constraints. This language revealed suitable to encode a large body of knowledge in a rather natural way, using classical preconditions/effects planning information but also additional operational knowledge in a formalism similar to HTN, but including programming language constructs (loops, conditional branching) and variables. This common representation ensures seamless transitions between modules. However, this raises the problem of the logical consistency between the various OP fields used by these modules. Part of this checking may be done using purely syntactic approaches, but other will require more formal logical techniques. Nevertheless, Propice-Plan provides means to avoid dangerous states with constraint OPs, which ensure that any conflict state is detected at execution time, and is not planned explicitly by the $\mathcal{A}m$ and the $\mathcal{P}m$.

In the future, we intend to study how to take some additional reasoning capabilities into account. At this point, Propice-Plan handles explicit operations on dates and durations, and wait / rendezvous synchronisations, but it is unable to manage time constraints propagation as described in [Ghallab and Mounir-Alaoui, 1989]. Similarly, better hypothetical reasoning techniques are needed to handle conditional branching for anticipation, and dynamic CSP techniques to cope with more general constraints. In addition, we intend to improve the current implementation, which already runs on board our mobiles robots, to use it on site for the blast furnace application (Propice-Plan runs under Solaris, VxWorks and Linux). In both application, hundreds of OPs have been encoded and are used by the $\mathcal{E}m$. The plans produced by the $\mathcal{A}m$ shows a real gain over the generic plans (eg. by preventing a furnace shutdown delay). Moreover, the $\mathcal{P}m$ is used in the robot application to synthesise multi robots plans [Botelho, 1998] and in the furnace application to handle the pipes network configuration.

References

[Alami *et al.*, 1998] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *International Journal of Robotics Research*, 17(4):315–337, April 1998.

[Beetz and McDermott, 1994] M. Beetz and D. McDermott. Improving Robot Plans During Their Execution. In *Proceedings of AIPS94*, June 1994.

[Blum and Furst, 1997] A. Blum and M. Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90:281–300, 1997. See also: http://www.cs.cmu.edu/~avrim/graphplan.html.

[Bonasso *et al.*, 1997] R.P. Bonasso, R.J. Firby, E. Gat, D. Kortenkamp, D.P. Miller, and M.G. Slack. Experiences with an Architecture for Intelligent, Reactive Agents. *Journal of Experimental and Theoretical AI*, January 1997.

[Botelho, 1998] S. Botelho. A distributed scheme for task planning and negotiation in multi-robot systems. In *Proceedings of ECAI*, 1998.

[Currie and Tate, 1991] K. Currie and A. Tate. O-Plan: the Open Planning Architecture. *Artificial Intelligence*, 52:49–86, 1991.

[Firby, 1994] R. J. Firby. Task Networks for Controlling Continuous Processes. In *Proceedings of the Second International Conference on AI Planning Systems, Chicago IL*, June 1994.

[Geffner and Bonet, 1998] H. Geffner and B. Bonet. High-Level Planning and Control with Incomplete Information Using POMDPs. In *Proceedings AIPS-98 Workshop on Integrating Planning, Scheduling and Execution in Dynamic and Uncertain Environments*, 1998. See also: http://www.ldc.usb.ve/~hector.

[Ghallab and Mounir-Alaoui, 1989] M. Ghallab and A. Mounir-Alaoui. The Indexed Time Table Approach for Planning and Acting. In Rodriguez and Seraji, editors, *Procs. NASA Conference on Space Telerobotics*, volume 5, pages 321–332. JPL Publications 89-7, Feb. 1989.

[Guéré and Alami, 1999] E. Guéré and R. Alami. A Possibilistic Planner That Deals With Non-Determinism and Contingency. In *Procs. IJCAI*, 1999.

[Ingrand *et al.*, 1992] F. F. Ingrand, M. P. Georgeff, and A. S. Rao. An Architecture for Real-Time Reasoning and System Control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering*, 7(6):34–44, December 1992.

[Ingrand *et al.*, 1996] F. F. Ingrand, R. Chatila, and R. Alami F. Robert. PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots. In *IEEE International Conference on Robotics and Automation, St Paul, (USA)*, 1996.

[Kambhampati, 1999] Subbarao Kambhampati. Improving Graphplan's Search with EBL & DDB Techniques. In *Procs.IJCAI*, 1999.

[Koehler *et al.*, 1997] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending Planning Graphs to an ADL Subset. In *Proceedings of European Conference on Planning*, 1997. See also: http://www.informatik.uni-freiburg.de/~koehler/ipp.html.

[Kushmeric *et al.*, 1995] N. Kushmeric, S. Hanks., and D. Weld. An Algorithm for Probabilistic Planning. *Artificial Intelligence*, 2:239–286, 1995.

[Levinson, 1994] R. Levinson. A General Programming Langage for Unified Planning and Control. *Artificial Intelligence*, 76:281–300, 1994.

[McDermott *et al.*, 1998] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL – The Planning Domain Definition Langage. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, October 1998. Available on request to drew.mcdermott@yale.edu.

[Myers, 1998] K. L. Myers. Towards a Framework for Continuous Planning and Execution. In *Proceedings of the AAAI Fall Symposium on Distributed Continual Planning*, 1998. See also: http://www.ai.sri.com/~cpef.

[Nau *et al.*, 1999] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. SHOP: Simple Hierarchical Ordered Planner. In *Procs. IJCAI*, 1999.

[Nebel *et al.*, 1997] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring Irrelevant Facts and Operators in Plan Generation. In *Proceedings of European Conference on Planning*, 1997. See also: http://www.informatik.uni-freiburg.de/~nebel.

[Pryor and Collins, 1996] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Artificial Intelligence Research*, 1996.

[Wilkins *et al.*, 1994] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and Reacting in Uncertain and Dynamic Environments. *Journal of Experimental and Theoretical AI*, 6:197–227, 1994. See also: http://www.ai.sri.com/people/wilkins/papers.html.

[Yang, 1997] Q. Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach*, pages 163–171. Springer-Verlag Publisher, 1997.