[16] W. Swartout. DARPA Santa Cruz Workshop on Planning - Workshop Report. *The AI Magazine*, 9(2):115–130, Summer 1988.

[17] R. Wilensky. Meta-Planning: Representing and Using Knowledge About Planning in Problem Solving and Natural Language Understanding. In A. Collins and E. Smith, editors, *Readings in Cognitive Science. A Perspective from Psycology and Artificial Intelligence*. Morgan Kaufmann, 1979.

[18] D. E. Wilkins. *Practical Planning: extending the classical AI planning paradigm*. Morgan Kaufmann, San Mateo, 1988.

[3] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of Theorem Proving to Probl em Solving. *Artif. Intell.*, 2(3-4):189–208, 1971.

[4] M. Georgeff. An embedded reasoning and planning system. In J. Tenenberg, J. Weber, and J. Allen, editors, *Proc. from the Rochester Planning Workshop: from Formal Systems to Practical Systems*, pages 105–128, Rochester, 1989.

[5] M. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proc. of the 6th National Conference on Artificial Intelligence*, pages 677–682, Seattle, WA, USA, 1987.

[6] F. Giunchiglia, P. Traverso, A. Cimatti, and L. Spalazzi. Programming planners with flexible architectures. Technical Report 9112-19, IRST, Trento, Italy, 1991. Revised version entitled: "Tactics: extending the notion plan" presented at the ECAI-92 workshop Beyond Sequential Planning (Vienna, August 1992).

[7] M.J. Gordon, A.J. Milner, and C.P. Wadsworth. *Edinburgh LCF - A mechanized logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer Verlag, 1979.

[8] K. J. Hammond. Explaining and Repairing Plans that Fail. *Artif. Intell.*, 45(1-2):173–228, 1990.

[9] B. Hayes-Roth. A Blackboard Architecture for Control. *Artif. Intell.*, 26:251–321, 1985.

[10] J. E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artif. Intell.*, 33(3):1–4 64, 1987.

[11] M. J. Schoppers. Universal plans for Reactive Robots in Unpredictable Environments. In *Proc. of the 10th International Joint Conference on Artificial Intelligence*, pages 1039–1046, 1987.

[12] R. Simmons. Concurrent planning and execution for a walking robot. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2086–2091, Sacramento, CA, 1991.

[13] L. Spalazzi, A. Cimatti, and P. Traverso. Implementing planning as tactical reasoning. In *Proc. AIS'92, AI Simulation and Planning in High Autonomy Systems Conference*, pages 80–85, Perth Australia, 1992. IEEE Computer Society Press. Revised version of Technical Report 9104-23, IRST, Trento, Italy.

[14] M. J. Stefik. Planning and Meta-Planning. *Artif. Intell.*, 16:141–169, 1981.

[15] L. Stringa. An integrated approach to artificial intelligence. Technical Report 9012-11, IRST, 1990.

is performed. Real time systems must respond immediately to external events, but complex and intelligent systems also need planning ahead.
MRG provides the ability to combine and integrate different planning techniques uniformly in one system by means of introspective planning. As far as we know, this approach is new and has never been proposed before. The idea of metaplanning is of course not new, see for instance MolGen [14] and [9, 10, 17]. However, none of these systems and approaches provides a metalevel programming language to reason about all the planning activities rather than only plan generation (like plan execution and monitoring, failure handling, information acquisition and reacting to external stimuli). All the activities of MRG are represented within MRG itself. We call this ability, *introspective planning* (to distinguish it from metaplanning). Introspective planning allows us to represent, alternatively use and combine different planning paradigms.
In PRS [5, 4], plans (called KAs) describe how certain sequences of actions and tests may be performed to achieve given goals or to react to particular situations. Metalevel KAs encode various methods for chosing among multiple applicable KAs. They provide a high amount of flexibility in forming plans. The same amount of flexibility is provided in MRG by tactics. Concerning the issue of introspective planning, in PRS, some of the fundamental internal PRS activities (like plan execution) are not explicitly represented by KAs. For this reason, classical plan generation has to be achieved by simulating execution [4] [pag. 125]. On the other hand, classical plan generation is one of the possibilities provided by MRG.
At the moment, we are experimenting with MRG in a complex large-scale application under development at IRST [15]. Different systems, like reactive sensor based systems for robot navigation, speech recognition and dedicated planning systems are integrated to create an intelligent robot able, among other things, to navigate in an unpredictable environment, to exchange information with people, to find persons in IRST and to help them. MRG is used to develop the central module that coordinates and activates all the other modules of the system. Most of the examples in this paper have been taken from this application. Introspective planning has turned out to be extremely useful and is being tested considerably by this application. At the moment asynchronous and parallel events are managed by primitive tactics. We have recently begun to examine extensions that allow parallel and asynchronous processes to be activated and controlled by proper constructs (new tacticals) in the MRG language.

References

[1] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[2] E.H. Durfee and V.R. Lesser. Incremental Planning to Control a Blackboard-based Problem Solver. In *Proc. of the 5th National Conference on Artificial Intelligence*, Philadelphia, PA, 1986.

tactic plans for moving the robot to the door of the office where the desk is located [`(door-loc-of (office-of desk-loc)`]. The plan to reach the desk location `(exec-path (plan-route-to desk-loc))` is then deferred after the planner knows whether the door of the office is opened [`(door-open?)`].
Reactive planning systems provide the ability to react immediately to environmental changes. In reactive systems, plan formation can be seen as the interpretation of an external stimulus plus the selection of an adequate precompiled reaction. Plan execution is the reaction itself. Failures are treated like any other stimulus, thus replanning is not needed. In `MRG` a reactive system can be represented by the following tactic:

```
(deftac reactive-loop ()
 (then (exec (select-reaction (get-stimulus)))
   (reactive-loop)))
```

where `reactive-loop` is a recursive tactic implementing a reactive loop, `get-stimulus` is a primitive tactic linked to a sensor controller and `select-reaction` selects an adequate reaction to the stimulus.
So far we have shown how `MRG` can be used to implement different planning techniques. But `MRG` can go "beyond the single planning paradigm". Different planning techniques can be alternatively used and combined according to the particular application requirements. As a simple example of this ability, we show how to implement a control mechanism that intermixes a reactive and a classical planning technique. For instance, we can plan ahead whenever no immediate reaction to an external event can be performed by our reactive system (*e.g.* in the case the external event is the request of a high level task that needs some search):

```
(deftac mixed-planning ()
 (let (input (get-stimulus))
  (then (exec (orelse (select-reaction input)
                      (plan-for input)))
        (mixed-planning))))
```

6 Related work and conclusions

Most of the existing planners implement a single planning paradigm. They may be well suited to solve some classes of problems, but it may be difficult to use them in complex real world large-scale applications, where different requirements have to be fulfilled by one system. For instance, classical planners have often been shown to be inadequate for highly unpredictable and dynamic environments; case-based planning has problems dealing with unpredictable domains; conditional planning cannot avoid the explosion of possible worlds if many conditions exist; deferred planning may be a solution for unpredictable domains but are perhaps not always suited to highly dynamic environments in which the world may change while plan generation

execution succeeds or no plan can be generated anymore. In MRG a simple example of a classical planner can be implemented by the following tactic:

```
(deftac classic-planner (goal)
 (let (plan (plan-for goal))
  (orelse (exec plan)
   (then (modify-planner-status)
         (classic-planner goal)))))
```

where plan-for and exec are a plan formation and a plan execution tactic respectively. modify-planner-status updates the planner internal representation of the world depending on the failure that occurred.
In *case-based planners*, plan generation is performed by using and debugging known plans. Once a planner has constructed a useful plan, the result is stored for later use. An example of case-based planning is shown by the following tactic[3]:

```
(deftac case-based-planner (goal)
 (exec (orelse (retrieve-tac goal)
               (learn-tac (plan-for goal)))))
```

where retrieve-tac is a plan generation tactic that searches for and debugs a tactic in a library and learn-tac is a tactic that updates the library and returns its argument (the plan generated by plan-for).
The plan formation phase of *conditional planning* involves "generating alternative plans for each possible outcome or world state, including an appropriate test in the plan to choose from the alternatives" [16]. Conditional plans can be obviously expressed within the MRG language by means of the tactical if.
Deferred planning systems can defer plan generation until some information is available by interleaving plan formation and plan execution. Within MRG it is possible to interleave plan formation and execution to acquire information at execution time as shown by the following example.

```
(deftac deferred-goto (desk-loc)
 (then
  (exec-path (plan-route-to (door-loc-of
                      (office-of desk-loc))))
  (if (door-open?)
   (exec-path (plan-route-to desk-loc))
   (then (open-door)
         (exec-path
            (plan-route-to desk-loc))))))
```

The tactic deferred-goto is a strategy to let the robot reach a desk location in an office [desk-loc]. Rather than planning for going to the desk location, the

[3]For simplicity, in this and in the next example, we do not consider replanning and recursively calling the planner itself.

also be very complex: for instance it could represent a full blown classical planner. Plan formation tactics can be either primitive or defined tactics. A primitive plan generation tactic represents a *fixed* process searching over tactics to find a solution to the given goal and is not modifiable by the user according to the application requirements. In example 1 in section 3, `plan-route-to` is a primitive plan formation tactic. A defined plan formation tactic is built from primitive ones. Its definition is accessible by the user and is modifiable according to the application domain.
Plan execution tactics: planning systems, given a plan that has been generated by the plan formation activity, execute the plan carrying out actions in the real world. In `MRG`, plan execution activities are represented by "execution tactics", whose arguments are tactics. An execution tactic, when interpreted, executes the argument tactic. In case of success it returns the result of the execution, otherwise it fails. In example 1 in section 3, `exec-path` is an execution tactic.
Replanning tactics: replanning is a way to react to failure. One of the arguments of a replanning tactic has to be of type goal. A replanning tactic returns an object of type tactic. In example 2, we show a replanning tactic that is currently implemented.

Example 2 : consider the plan formation tactic `plan-route-to` in example 1 in section 3. If the execution of the plan built by `plan-route-to` fails, then the tactic `replan-route-to` can be used to react to failure by replanning.

```
(deftac replan-route-to (goal-loc old-plan)
 (let (current-loc (query-position))
  (if (is-out-of-route? current-loc old-plan)
      (plan-route-to goal-loc)
      (then (modify-map current-loc old-plan)
            (plan-route-to goal-loc)))))
```

This tactic checks, by asking for information to the sensors [`query-position`], whether the robot halted along the route that has been planned or the robot lost the route [`is-out-of-route?`]. In the latter case, a new plan is generated from the current position [`(plan-route-to goal-loc)`]. Otherwise, an alternative path is generated that avoids going through the area that caused the robot to halt.

Acquisition tactics: these tactics acquire information from the real world, usually through the use of sensor controllers. For instance, `query-position` in example 2 and `user-interface-confirm` in example 1 are acquisition tactics.

5 Beyond the single planning paradigm

All the planning paradigms differ depending on how they perform, activate, combine and control the various planning activities, some of them described in section 4.
For instance, most *classical planners* perform a phase of plan generation followed by plan execution. In case of failure during plan execution, they replan again until the

with the intuition that after the failure of an action, some alternatives can be tried, and the overall plan may succeed in any case [6]. Accordingly, `orelse` may be thought of as a retrial point: If τ_1 fails, then τ_2 is executed and the overall tactic succeeds or fails according to τ_2. If τ_1 succeeds, then the overall tactic succeeds.
Recursive plans can be defined in `MRG`. `MRG` features a definition mechanism through the construct `deftac`. `deftac` introduces new tactic identifiers. It supports definitions of recursive tactics. In example 1 we show a simple defined tactic. Examples of recursive tactics are shown in section 5.

Example 1 : consider the (simple) problem of sending a mobile robot to a given place in a building (`location`) only if the operator agrees on the path generated by the planner. A defined tactic which can perform this function is the following:

```
(deftac goto-ask-user (location)
 (let (path (plan-route-to location))
  (if (user-interface-confirm path)
      (exec-path path)
      (user-interface-inform))))
```

The tactic above is a plan, built by using the tacticals `let` and `if`. The tactic identifier `goto-ask-user`, defined by the `deftac` construct, plans for a navigation path [`plan-route-to`] to the location represented by the variable `location`, records the route in the variable `path`, asks the user to confirm the path [`(user-interface-confirm path)`] and, only if the answer is positive, actually executes the plan to move the robot along the path: `(exec-path path)`; otherwise information is displayed through the user-interface module: `(user-interface-inform)`.

4 Introspective planning: representing `MRG` within `MRG`

The language defined in section 3 is expressive enough to represent the `MRG` basic planning activities. These planning steps are explicitly represented in the language of `MRG`. Thus they can be programmed and flexibly modified [13]. All the various planning activities are uniformly represented within `MRG` by means of tactics. Some of them are listed below.
Plan generation tactics: in planning systems, plan generation (also called plan formation[2]) is the generation of a plan to achieve a given objective. In `MRG`, plan generation activities are represented by tactics whose arguments are goals and that return tactics. The simplest plan formation tactic we can think of is a function that, given a goal, returns the corresponding tactic in a goal-tactic table (this idea resembles in some way the reactive approach). Of course, a plan generation tactic can

[2]Plan generation is sometimes simply called "planning". By "planning activities" we mean all the operations that a planning system performs and not only plan generation, *e.g.* plan execution, monitoring, reaction to events etc.

or failure, the interpretation of the tactic is said to succeed or fail. For instance, the interpretation of the primitive tactic `(follow-wall)` activates a reactive system moving a robot along a wall in a building; `(plan-route-to room-100)` is a primitive tactic planning for a navigation path to the location represented by the constant `room-100`. `(exec plan)` is the introspective primitive tactic representing the `MRG` execution of the plan `plan`.
A primitive tactic is an atomic object in the `MRG` language. The corresponding action is fixed. For instance, since `plan-route-to` is primitive, it is a fixed plan formation mechanism. Notice that this is what usually happens in traditional planning systems: the plan generation procedure is fixed in the system code.
In `MRG`, the various planning activities can be flexibly controlled by means of more complex tactics which can be obtained by combining primitive tactics in different manners.

Definition 2 (Tactic)

- *A primitive tactic is a tactic.*
- *Let* τ_1 *and* τ_2 *be tactics. Then* `(then` τ_1 τ_2`)` *is a tactic.*
- *Let* x *be a variable. Let* τ_1 *and* τ_2 *be tactics. Then* `(let (`x τ_1`)` τ_2`)` *is a tactic.*
- *Let* τ_1, τ_2 *and* τ_3 *be tactics. Then* `(if` τ_1 τ_2 τ_3`)` *is a tactic.*
- *Let* τ_1 *and* τ_2 *be tactics.* `(orelse` τ_1 τ_2`)` *is a tactic.*
- *Let* `(`t_i $c_1 ... c_n$`)` *be a primitive tactic. The expression obtained by replacing any of* c_i *with either a variable or a tactic is a tactic.*

`then`, `let`, `if` and `orelse` are symbols of the `MRG` alphabet. They are called *tacticals* and map tactics onto tactics.
`then` is the tactical for the construction of sequential tactics; the argument tactics are intended to be executed in the given sequential order. If τ_1 fails, then τ_2 is not executed and the overall tactic fails. If τ_1 succeeds, then τ_2 is executed and the overall tactic succeeds or fails according to τ_2.
`let` is used to capture a particular aspect of the world (the result of the evaluation of τ_1) and eventually use the information stored in the variable x later on as argument to other actions. The `let` tactical is needed because the evaluation of a tactic can be substantially different in different situations. For instance, `let` can be used to store information processed by the sensor controllers of a robot and that are likely to change (like the position of a moving object over time). A let tactic fails if and only if one of τ_1 and τ_2 fails.
The `if` tactical builds conditional tactics. τ_1 is a tactic whose evaluation returns, when it succeeds, a truth value. When a conditional tactic is executed, τ_1 is executed first; if it succeeds, τ_2 or τ_3 are executed according to the result. A conditional tactic fails if one of the executed tactics fails.
The tactical `orelse` is the primitive operation for failure detection and for specifying reaction to failure. `orelse` is in a sense the most powerful tactical. If we consider tactics built only using the tacticals defined so far, every time the execution of a primitive tactic fails the overall tactic fails too. This is in complete disagreement

2 An overview of MRG

MRG provides a programming language for implement planning systems. Expressions within the language can be defined to represent

- the entities that a planning system usually reasons about, like goals, external events and actions;
- the MRG planning activities, like plan generation and execution, replanning, reaction to events in and information acquisition from the external environment;
- complex plans of actions and MRG planning activities involving a variety of control constructs (*e.g.* recursion, conditional expressions and sequences).

These expressions have a given type. For instance, objects of type *goal* represent goals to be achieved. MRG planning activities and plans are represented by expressions of type *tactic*[1]. Tactics are programs in the MRG language: they represent, control and execute actions in the real world, plans of actions and various activities of a planning system. For instance, a tactic can specify a set of actions to be executed in a given order, depending on some conditions (*e.g.* a particular external situation or a failure that occurred). A tactic may specify that a plan must be generated for a given goal, or that some action must be executed before, after or during plan formation. In a tactic, we can specify that a particular plan must be executed or that, when the execution fails, an alternative plan has to be tried, or that some replanning for a given goal must be performed.

3 A programming language for introspective planning

In this section we define the MRG language. Constants and tactic identifiers are symbols of the language. Tactic identifiers represent executable actions. Constants represent entities over which actions are executed.

Definition 1 (Primitive tactic) *Let $c_1, ..., c_n$ be constants. Let t_i be an n-ary tactic identifier. Then $(t_i \;\; c_1 ... c_n)$ is a primitive tactic.*

Each tactic identifier has associated an executable action. The concept of executable action is extented with respect to the usual notion. Actions may be executable in the external world (*e.g.* moving a robot) or may be executable as introspective MRG planning activities (*e.g.* generating or executing a plan to achieve a goal). In the latter case, actions are the actual MRG code subroutines. Actions are, in general, pieces of executable code, that, when executed, may fail or succeed. When primitive tactics are interpreted, the corresponding actions are executed: according to their success

[1]The term tactic is borrowed from ML [7], a procedural metalanguage for the specification of theorem proving strategies. There actually exist some similarities between the two languages, but there are rather strong conceptual and technical differences, mainly due to the different domains of application.

systems have been shown to be more adequate for implementing real time systems than classical planners. On the other hand, classical planners seem to be well suited for implementing problem solvers in which "off-line" reasoning and planning ahead is heavily required.
The work presented in this paper is based on the idea that real world large scale applications and intelligent systems can hardly be developed according to only one of the many existing planning paradigms. On the contrary, more than one of the different approaches and techniques have to be used and integrated [12]. For instance, most of real world systems need a planning ahead phase to anticipate predictable events and situations, they need to interleave planning and execution when information is not available and they need to respond immediately to environment changes. Furthermore, the best way to perform and combine these tasks depends heavily on the particular application domain.
For this reason, `MRG`, the system described in this paper, rather than implementing one planning technique, provides the user with the ability to implement a system that alternatively uses, combines and integrates various planning techniques depending on the application domain. This is achieved by extending the notion of plan. In `MRG`, a plan consists not only of (representations of) the actions executable in the external world, but also of representations of the *internal planning activities,* i.e. *plan generation, monitoring and execution, replanning, information acquisition from the real world, reaction to external changes and so on.* According to this extended idea of plan, a plan itself can describe when and how to generate a plan for a given goal and when and how to respond immediately to external stimuli. It can describe how and under which conditions to execute a generated plan and to acquire information from the external environment. It can state whether it is better to replan a new course of actions after a failure occurred or to execute directly some failure handling strategy. This opens up the possibility of performing *introspective planning*, that is *reasoning about all the planning activities of the system.* Different and complex controls and combinations of these planning activities can be obtained by executing different `MRG` plans. Within `MRG`, it is thus possible to build, for instance, classical, deferred, conditional, case-based and reactive planners. More importantly, it is possible to build planning systems that combine these different planning techniques in a single system according to the application domain.
Our paper is organized as follows. In section 2 we give a brief overview of `MRG`. In section 3, the `MRG` language to express plans is defined. In section 4 we show how the various planning activities of `MRG` can be explicitly represented by expressions of the `MRG` language. Section 5 shows how these expressions can be combined to implement and integrate different planning techniques. In Section 6 some related work is considered and some conclusions are given.

Beyond the single planning paradigm: introspective planning*

Paolo Traverso[1], Alessandro Cimatti[1], Luca Spalazzi[2]

[1]IRST - Istituto per la Ricerca Scientifica e Tecnologica,
38050 Povo, Trento, Italy

[2]Istituto di Informatica, University of Ancona,
Via Brecce Bianche, 60131 Ancona, Italy

Abstract

Real world large scale applications require planning systems that combine different planning techniques (like planning ahead, deferred planning and reactive planning). In this paper we present a system (called `MRG`) that provides this capability. In `MRG` plans, the planning activities of the planner itself (like plan generation and execution, information acquisition from the real world and reaction to external changes) are explicitly represented. Thus, a `MRG` plan can describe whether, when and how to perform these activities. As a consequence of this ability (that we call *introspective planning*), `MRG` can be used uniformly and flexibly to combine different planning techniques in one system according to the application requirements.

1 Motivations

A variety of approaches to planning have been proposed so far in the AI literature, for instance classical planning [3, 18], case-based planning [8], conditional planning [11], deferred planning [2], reactive systems [1, 5]. Each of these different approaches has been successfully applied to certain classes of problems. For instance, reactive

*This work has been conducted as part of MAIA, an integrated AI project under development at IRST and has been done with the partial support of the Italian National Research Council (CNR), Progetto Finalizzato Robotica (Special Project on Robotics). Fausto Giunchiglia has supervised the development of the whole project. He provided many of the underlying intuitions. The Mechanized Reasoning Group at IRST is also thanked.

ISTITUTO PER LA RICERCA SCIENTIFICA E TECNOLOGICA
I 38100 TRENTO – LOC. PANTÉ DI POVO – TEL. 0461–314444
TELEX 400874 ITCRST – TELEFAX 0461–302040

BEYOND THE SINGLE PLANNING PARADIGM: INTROSPECTIVE PLANNING

P. Traverso, A. Cimatti, L. Spalazzi

April 1992
Technical Report # 9204-05

Published in *Proceedings ECAI-92, 10th European Conference on Artificial Intelligence*, Vienna, 3-7 August, 1992, pp. 643-647.

ISTITUTO TRENTINO DI CULTURA