Reconfiguration and Replanning for Robust Execution of Plans Involving Continuous and Consumable Resources

Enrico Scala

Ph.D. Thesis
Supervisor: Prof. Pietro Torasso

Universita' degli Studi di Torino
Dipartimento di Informatica
XXV Ciclo :: Scuola di Dottorato in Computer Science

April 2, 2013

Contents

List of Figures

List of Tables

Acknowledgements

This thesis would not have been possible without the people I have knew throughout these three long years of the PhD course.

First of all I would like to express my deepest gratitude to Prof. Pietro Torasso, for his huge contribution in all the aspects of the thesis, for his availability in spreading his enormous knowledge on the Artificial Intelligence field, and for all the discussions we had during all my PhD. I very thank him for having introduced me in the AI.

I would like to thank the reviewers, Dott. Amedeo Cesta and Prof. Cees Witteveen for their helpful suggestions which helped to improve the quality of the thesis and to highlight the importance of the contributions.

I am very grateful to Dott. Roberto Micalizio for all the time he dedicated in approaching me to the problems and the discussions we had during this period of work, even if we often have had different points of view on the possible solutions:).

A big thank you to Massimo. He is such a big friend of mine, who helped me during the thesis writing phase, and not only. He is one of the reasons why this thesis is readable and, I hope, comprehensible. But he is also definitely the nightclubbing/sport/girl-mentor/holiday friend with who I spent the most of the time here in Torino.

I thank my family, and in particular, my mother Francesca and my father Antonio who supported me economically and emotionally since I was born. Without them I would not be who I am. Thank you.

I thank my girlfriend Sara who always believes in me (I do not know why sometime), and all the friends who accompanied me during this travel: Marco DM, Mariano, Jennifer, Alberto, Giorgio, Michele, Seby, Agata, Cristiano and so forth. Last but not least, Fabio Varesano, who is not here anymore but with who I spent the most of the free-time at the department this last year. I dedicate this thesis to his memory.

Chapter 1

Introduction

The implementation of agents with a high degree of autonomy is of a paramount importance in the modern society. Autonomy is indeed desirable in many fields as service robotics, gaming, web applications, space missions, virtual assistants and so forth. In particular, the capability of automated planning is key and even mandatory when the agent is in charge to perform goal-directed tasks.

The automated planning is a traditional branch of the Artificial Intelligence and has been studied since the introduction of STRIPS ([41]) where the propositional (nowadays called classical) planning has been introduced. Planning concerns the reasoning ability of finding a course of actions to achieve a set of goal conditions starting from a particular description of the state.

However, while for deterministic and predictable environments (e.g. mainly puzzle and toy problems) high levels of autonomy are achievable still with the classical setting, acting in the real world may be a difficult task for an agent either software or robotic. Real-world scenarios, in fact, are in general weakly predictable and highly dynamic, and the agent may have just a partial knowledge of the environment where it is operating. For instance, a planetary rover has to operate in a hazardous and not fully known environment where a number of unpredictable events may occur.

This means that unexpected contingencies may arise at any step of the execution and some assumption made at planning time may turn out to be wrong. It is not thus surprising that the planning community has recently focused on the problem of *robust execution* in real-world domains.

A first methodology to cope with the problem consists in anticipating, at planning time, all the possible contingencies that might occur during the actual execution of the plan. The result of the planning phase is therefore a conditional plan [26], i.e. a plan where alternative courses of actions are possible depending on certain conditions of the environment. During the execution phase, the agent guided by its sensing actions is able to select a feasible branch of its plan that

leads to the goal. A similar methodology has been proposed in [27]; in this case, conditional plans are built to guarantee that the agent satisfies temporal constraints on the achievement of its goals.

These approaches are successful when it is possible to anticipate all the contingencies at planning time (i.e., off-line). When contingencies are not fully predictable or the agent is prevented to have complete knowledge of the environment, a continual revision of the plan is necessary. For this reason it is going to become more and more widespread the adoption of the continual planning paradigm. Initially introduced in [34], and adopted (*de facto*) by most of the architecture for autonomy that have been developed (e.g. Alami's architecture [5], GOAC [19], CLARATY [76], STANLEY [96]) a continual planning agent can interleave planning and execution all along the task assigned. Instead of anticipating the contingencies at planning time, the agent can exploit the knowledge acquired during the execution and face the contingencies once they are actually detected[1].

Many works [97, 60, 2, 44] have recently proposed to enhance robust plan execution via a continual revision of the plan. Basically, these works suggest to stop the plan execution as soon as some unexpected situation is encountered attempting then to repair the broken plan either via replanning from scratch or via plan adaptation. Note that, plan adaptation mechanisms usually substitute a portion of the original plan with a new plan segment synthesized on-the-fly to overcome erroneous situations. The idea is that the new plan segment overcomes the wrong conditions that have stopped the previous execution, so that the agent can resume the execution of what remains of the original plan.

The detection of the inconsistency is managed by *simulating* the impact of the plan on the state of the system, while the repair step assumes that reusing part of the old solution is more convenient than replanning from scratch. Even if the adaptation process is considered hard as much as planning from first principle ([75]), these approaches demonstrated that the repair can work very well in practice.

In this thesis we address the robust plan execution from a different perspective: to improve as much as possible the robustness of the execution we propose a multi-level strategy which extends the replanning with a reconfiguration mechanism. The result is a system that keeps efficiency and reactivity in a unified architecture.

We start from the observation that real-world scenarios include both reusable and consumable resources. Thus, a plan has not only to achieve a set of goals, but it has also to conform to strict constraints on the amount of resources to be used by the agent. For this reason, it is necessary to explicitly model the

[1] The problem of robust plan execution is still harder when the observability is not complete. In those contexts a step of plan diagnosis is mandatory for the effectiveness of the agent. For details see [29],[87],[62],[54],[55]

(expected) profile of resource consumption during the execution of any action. We thus model consumable resources as numeric fluents, which are explicitly mentioned in the preconditions and effects of the action templates, and in the requirements of the task.

We also observe that in many real-world domains it is possible to identify actions performing the same task (i.e., obtaining the same effects), but requiring different configurations of the agent. These actions share the same qualitative objectives, while they differ in the way they are performed. Typically, different configurations have different resource profiles. We therefore propose to group these subsets of similar actions by introducing the notion of *multi-modality action* (MMA). An MMA represents in a compact form alternative ways, or alternative *execution modalities* to accomplish a task. In planning terms we would say that all the execution modalities of a given MMA reach the same set of propositional effects. However, since they are characterized by specific resource consumption profiles, they differ in their numeric preconditions and effects.

MMA are organized and instantiated in Multi Modality Plans (MMP) and as we will see such plans can be computed as a result of a generic Multi Modality Planning Problem. An MMP in the context of the plan execution can be managed by the Dynamic Modality Allocation Problem (DMAP), which in turn can have a computational representation in the Constraint Satisfaction Problem formalism (CSP). The CSP mechanism reached a high degree of maturity and has been recently applied in the context of planning. Nevertheless the current conversion are deeply investigated in the propositional fragment ([36],[59]). In this thesis we aim at specializing the translation for the numeric context without making any assumption on the particular solver to be used.

Relying on the notion of the MMPP and DMAP the thesis presents FLEX-RR (*FLexible EXecution via Reconfiguration and Replanning*), which is an extension of the on-line approaches based on replanning. The approach is inspired to the continual planning paradigm [15, 34] since it allows the agent to interleave (re)planning and execution.

The idea is that, while replanning is in some cases unavoidable, it may be an "excessive" reaction in many situations. FLEX-RR tries to limit the replanner's intervention by singling out those situations which could efficiently be resolved via (a simpler) reconfiguration phase. On the other hand, when plan reconfiguration fails or is not sufficient, FLEX-RR is able to start a replanning task.

FLEX-RR has many advantages. First of all, the reconfiguration is beneficial from the computational point of view, since reconfiguring action modalities is not as expensive as synthesizing a new plan from scratch. Secondly, since FLEX-RR encompasses both reconfiguration and replanning, it can deal with those situations solvable via replanning but not via reconfiguration. Finally, via the

reconfiguration of action modalities we are able to keep the repaired plan highly *stable*. This is important when the agent is situated in a multi-agent setting.

Based on the notion of the MMA, the thesis proposes a further level of control, namely the Active Supervisor (ActS). The main objective of ActS is to avoid (at least in some cases) the occurrence of *action* failures *during* the same action execution, improving as a consequence the robustness of the whole plan execution. More precisely, to avoid failures ActS aims at assuring that (i) the action execution does not prevent the safety of the system, (ii) the action achieves the expected effects (including the way in which resources are employed). For these reasons, ACTS plays the role of a short term supervisor, while FLEX-RR a long term one.

In order to improve the reasoning about the numeric fluents the plan has to deal with, the thesis proposes the notion of Numeric Kernel, which is a generalization of the propositional kernel presented for the STRIPS language [41]. Analogously to the propositional case, the Numeric Kernel provides a way for focusing the agent attention only on those information which are actually relevant for the planning problem at hand.

The Numeric Kernel will be exploited to provide an heuristic guidance for ActS by means of the *safe execution modalities* notion. The introduction of such a notion allows to guide the reactive mechanism of ActS to prefer only configurations of action parameters that do not compromise the feasibility of the plan. As a consequence the cooperation between ActS and FLEX-RR becomes more effective.

The Numeric Kernel has been adopted also for two other enhancements of FLEX-RR: (i) the monitoring task for the detection of inconsistencies, (ii) the replanning mechanism with a more focused strategy.

The effectiveness and the innovations of the system will be made evident via a number of experiments that will be performed on three planning domains: the *Planetary Rover* domain, the *ZenoTravel* domain and the *DriverLog* domain. The experiments are aimed at evaluating:

- The competence of (various configuration of) FLEX-RR in handling unexpected contingencies.
- The competence of ActS in anticipating action failures.
- The efficiency in the tasks assigned to FLEX-RR. Particular emphasis will be given to the reconfiguration mechanism.
- The efficiency and the quality of plans produced by FLEX-RR with the replanning enhanchement provided by the *numeric kernel*.

The thesis has been partially developed within STEPS (Sistemi e TEcnologie Per lo Spazio), which was a three years long *R&D* project aimed at investigating

methodologies for the next Space Missions. The project involved a number of Industrial and Academic organizations and was leaded by Thales Alenia Space. The space exploration scenario allowed us to see concretely some of the issues regarding the robust execution and the autonomy in general.

1.1 Outline of the Thesis

The thesis is organized as follows:

- Chapter 2 reports the state of the art in the context of Robust Execution. The discussion reports three different perspective on the topic: *Architecture for Autonomous Systems and the Continual Planning Agent*, *Robust Plan Execution*, and *Robust Schedule Execution*.
- Chapter 3 introduces the basic formal framework of reference for the thesis. It proposes the model of Multi Modality Actions (MMA) as a means of describing actions which may be executed with different execution modalities.
- Chapter 4 formally introduces the Multi Modality Plans as the result of the Multi Modality Planning Problem and the Dynamic Modality Allocation Problem. Moreover it reports some interesting properties of the given formulations and a new notion of stability adapted for the Multi Modality Plans.
- Chapter 5 reports the Constraint Satisfaction Problem translation mechanism adopted for a computational representation of the Dynamic Modality Allocation Problem.
- Chapter 6 describes the FLEX-RR architecture and the algorithms used to supervise the Multi Modality Action Plans presented in Chapter 4.
- Chapter 7 reports and discusses the experimental results obtained by using different configurations of FLEX-RR.
- Chapter 8 introduces the Numeric Kernel notion as a tool for reasoning about plans involving numeric fluents. Moreover it discusses the benefit of the Numeric Kernel within the overall FLEX-RR-ACTS architecture.
- Chapter 9 describes the Action Supervision method employed for reconfiguring durative action on the fly, meaning while the action is still in execution.

Chapter 2

State of the Art on Robust Execution

Our approach proposes a methodology for the robust execution of a plan in *real-world-like* domains. In particular, the plan execution has to deal with the presence of consumable and continuous resources as time, space, energy and so forth.

In general, the actual plan execution may be threatened by the occurrence of unexpected contingencies and anomalies which may both prevent the agent to achieve the goals and violate the constraints on the resources. The thesis focuses hence on the study of techniques and strategies to be provided to an agent for the purpose of making the execution of its plan of actions as tolerant as possible.

In literature, the robust execution of tasks has been tackled by different perspectives. There is a variety of school of thought and a unified point of view explaining how the issue should be dealt did not arise yet. It is quite common to measure the effectiveness of an agent in performing a task as her ability in making predictions about the environment in which she is operating. Nevertheless, describing the evolution of environment is difficult to be modeled and even if a high degree of accuracy is achieved these models become easily intractable from the computation point of view. Therefore achieving robustness is a very tricky task. This Chapter aims at providing an interpretation of the different approaches to the problem of robust execution by breaking the discussion into three macro research areas.

Architectures for Autonomous Systems and Continual Planning. As a first point, we will look at the problem from a high level perspective. There is indeed a quite big amount of research in designing architectures and agents that are supposed to accomplish complex tasks in real world environments. The main purpose is to clarify the interdependences among the planning, the

execution and the sensing phases. The prevailing approach that will arise from this field is crucial for the comprehension of the spirit behind our proposal, and in particular in the multi-layered strategy of FLEX-RR and ACTS (see Chapter 6 and 9).

Robust Plan Execution. The second area deals with the problem of robust execution from a perspective close to the automated planning field. The robust execution of such plans has to guarantee that the causal structure is preserved and that the goals continue to be achievable all along the plan execution, that is, the plan has to be valid till the end of its execution.

Robust Schedule Execution. The third area of research sees the plan as a set of temporized durative events. Events can be in temporal relations among each other therefore the robustness is guaranteed once such relations remain valid during the plan execution. The temporal reasoning and hence Simple Temporal Network (STN) and/or Disjunctive Temporal Network (DTN) are the more widespread models at the basis of this class of approaches.

2.1 Architectures for Autonomous Systems and Continual Planning

The use of the agent paradigm (as for instance the BDI architectures [83]) is a widespread approach for the implementation and the development of autonomous systems.

In this perspective, among researchers of AI it is common that in order to develop an intelligent behavior, one of the key element in an agent is the ability of deliberation. That is, the agent should have the inference capability to reason on a situation and achieve a set of decisions that can be used to satisfy its wishes (or goals). Moreover, since the environment where the agent carries on its activities can be dynamic, some sort of reactive capabilities are necessary in order to not compromise the agent safety.

It is quite common that deliberation at an adequate level of abstraction can be performed by symbolic reasoning. However, the implementation of agents embodying real world applications (robotics, web-agents, autonomonic computing and so forth) requires a deeper attention since the process interconnecting the representation/reasoning level and the actual real world representation is quite complex. As indeed discussed in [100], on one hand the reasoning phase has to be able to efficiently produce decisions before they become useless; on the other hand, the process of transduction of the actual world in symbolic structures has to be as fast as necessary to be useful for the successive reasoning phase.

In a nutshell, in order to exhibit a real competence in solving tasks (and hence in order to being effectively intelligent), an agent based system should be

able to close the sense and reasoning loop in a reasonable amount of time and in the meanwhile reacting when necessary.

Due to skepticism on the efficiency of pure deliberative systems based on symbolic reasoning (e.g. STRIPS[41]), the problem of combining the deliberation and reaction has been initially solved by simply ignoring the deliberation capability and shifting towards pragmatic and simplistic solutions ([100]).

The main idea is that a purely reactive agent could be sufficient (or anyway necessary) to develop autonomous application. A reactive agent has a set of compiled rules encoding the complete and sufficient knowledge for acting. That is, the action selection is just a triggering process which relates the perception and the execution. In this perspective the agent is not requested to perform any form of complex reasoning for understanding which action should be selected, rather its behavior is a direct consequence of the surrounding environment. The intelligence of the agent in this case comes directly from the interaction with the environment. Example of such approaches are summarized by the subsumption architecture ([17]) and by [4].

To overcome the quite evident limitations of a purely reactive system (e.g. the lack of goal directed behaviors), one of the most influential and recent trend is the adoption of hybrid architectures involving both high level reasoning (deliberation) and fast reactive behaviors. Despite this perspective increases the difficulty in the implementation of the system mainly because of many different levels (and kinds) of representation, it turned out to be a promising approach; in fact many hybrid architectures have been employed for successful agent based applications, recently ([96], [76], [35] and [69]).

For this reason, in the next Section we will look at an example of a three tiered architecture. This model of agent well summarizes the basic schema of an hybrid approach; moreover it is becoming a standard *de facto* in a number of real world agent based applications.

Finally in Section 2.1.2, we report a Continual Planning approach developed for an agent acting in a context with limited knowledge of the environment. The Continual Planning is a quite young research area of the AI planning community which is gaining attention in the recent years. Since the main objective is to study the integration of the planning and the execution in a unified vision, it presents many analogies with the philosophy behind agent architectures.

2.1.1 Multi-layered Architectures

Most of hybrid architecture can be summarized by the architecture proposed by Alami ([5]) and the 3T architecture ([12]). The underlying idea is to encode the capabilities of the agent in a hierarchical way. Each layer reasons to a specific level of abstraction providing actions and receiving outcomes from the level below it, until actions and sensing in the real world are reached.

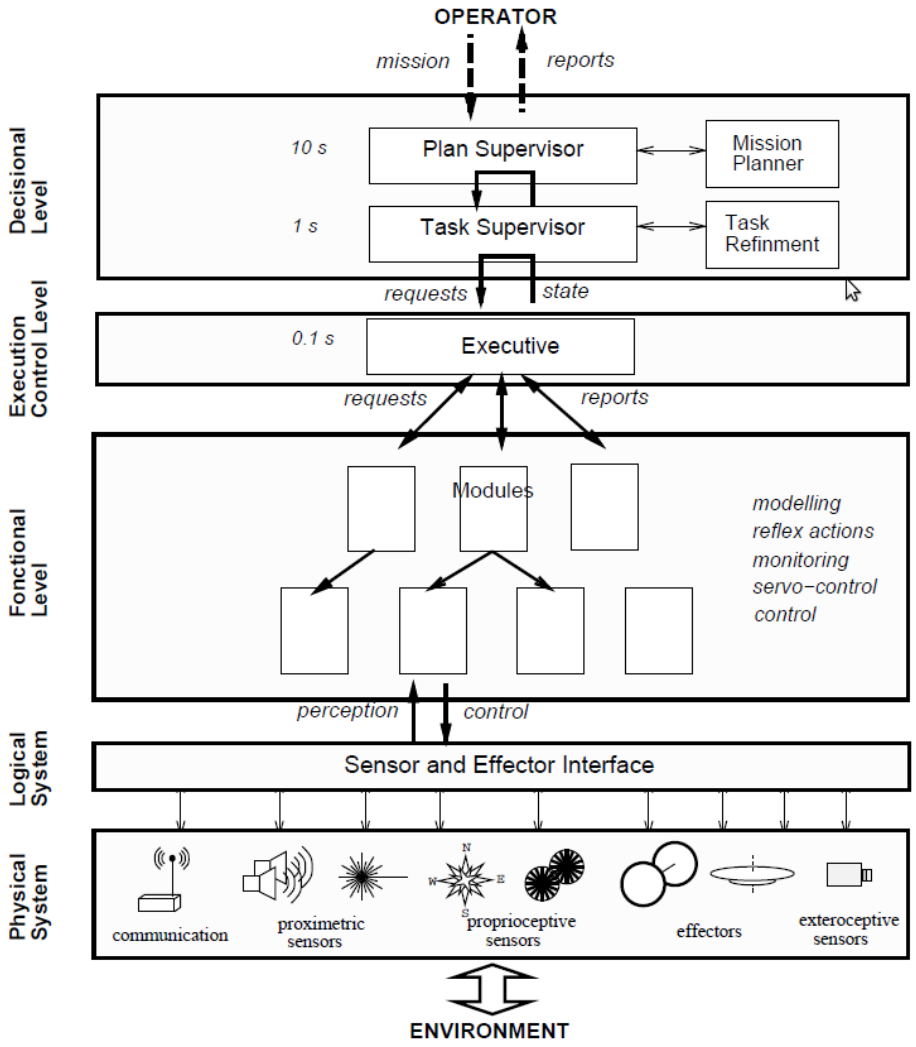


Figure 2.1: The Architecture for Autonomy presented by Alami et al. in [5]

More precisely (as reported by Figure 2.1), the most of the approaches summarizes these levels into three main macro layers, i.e.:

- Decisional level (Top of the Figure)
- Executive level (Middle of the Figure)
- Functional level (Bottom of the Figure)

The first one deals with deliberative and reactive abilities the agent should have producing plans and overseeing their execution. It may encompass more levels of abstractions for the same problem, each one consisting of a Planner and a Supervisor where the first provides the latter with the plan of actions to be executed. The Supervisor handles the execution capturing events and then allows the agent to exhibit reactive behaviors. Each couple of Planner and Supervisor produces a sequence of actions that becomes the sequence of problems handled by another couple assigned to an immediate lower abstraction of the problem. In other words the Supervisor sends its actions to the level immediately below, which in turns plans activities to achieve such actions. The delegation mechanism terminates when the action to be accomplished can be managed by the Functional Level.

The Functional Level (FL) consists of a certain number of modules and each of them is assigned to the resolution of a given task (e.g., Path Following, Vision, Manipulation and so forth). The FL interacts with the physical hardware (both for sensing and for acting) of the agent affording tasks (more or less complex) that are the basic actions taken into account by the Decisional level. Each task performed by the FL is pre-programmed and it may exploit one or more interacting modules. The FL is supposed to not have symbolic decision-making capabilities, rather it works at a lower level by interpreting signals and assuring competence in performing specific actions.

The Executive Level (EL) is defined as the pivot interface assigned to fulfill the gap between the symbolical/declarative and the numerical/procedural representation of the DL and FL respectively. Decisional level leads plans that manage high abstraction of the problem, for this reason it requires lower sensing commands frequency, differently by the Functional level, which has to constantly update information in order to handle strict control of low level-commands. Issues arising from difference in data flow rate between DL and FL are up to the Executive level, which is responsible of allowing, on one hand, the DL to obtain services from the FL and on the other hand, the FL to report results to the DL.

There are a number of works inspired to the organization reported above. For example in the space application the CLARATY architecture ([76]) has been used as framework for the implementation of the control mechanisms of the NASA mobile robots, which have been sent on Mars, i.e. Spirit and Opportunity. In CLARATY the functional layer encompasses the low level locomotion

control and the main perception capabilities; the executive layer is implemented by using the PLEXIL environment ([98]), whereas the decisional capability relies on the EUROPA system ([11]). Another example of successful applications of the multi tiered schema is the navigation control system developed by Thrun et al. ([96]), which won the 2005 edition of the DARPA competition (`http://archive.darpa.mil/grandchallenge05/gcorg/index.html`). In particular, in such a system the lower level is in charge of interpreting the information provided by the variety of cameras and sensors which equips a car (sensor interface layer); immediately on the top, the Perception layer transforms the information into a set of data representing the state of the system (position, trim, velocity and so forth). Such information are used by the planning and control layer which is in charge of computing and following a path. The architecture ends with user interfaces which allow the interaction between the human and the system.

It is worth noting that the proposal of such an architecture largely contributed in identifying the different roles played by agents that have to be as independent as possible. No concern however is highlighted regarding agents that may interact each other and it seems very hard to disassemble this architecture to allow cooperative tasks. Some solutions are reported in InterRRaP ([68]) and in [91]. In particular, Sellner et al. ([91]) presents an approach for a multi robot system where each robot is modeled as a common three tiered architecture; the cooperation is allowed among the different layers of the architecture. For instance the deliberative control of a robot A can provide the decisional capability for the functional layer of a robot B, which in turn supplies the necessary information about its vision of the environment to A. The idea is to optimize the overall computational power and share the knowledge of the environment by merging the pieces of information coming from robots different points of view.

2.1.2 A Continual Planning Agent

Multi tiered architectures for autonomy seem the most viable solution in the implementation of real world agents. However, from a theoretical point of view it is difficult to establish a clear semantic for the operations and the tasks to be executed. The problem arises from the difficulty to figure out an unified reasoning mechanism behind the multiplicity of modules involved in such layered structures. As a demonstration of this, most of the works reported above focuses on ad-hoc (and mainly robotic) solutions.

For the purpose of identifying a domain independent methodology, the continual planning paradigm has been proposed ([34]). This is an approach for the implementation of agents devoted to perform tasks in the real world environments, where a continual revision of the plan is necessary in order to account the

multiple and unpredictable contingencies that may occur. To this end, a continual planning agent is allowed to interleave planning and execution all along the task assigned. Precisely, the agent is supposed to have two main capabilities[1]. That is:

- the agent should be able to understand *when* interrupt or revise the execution (for example because the plan is not valid anymore);
- the agent should be able to decide *how* to recover from the impasse.

Of course, both steps have to be executed in a timely fashion to be useful.

Instead of relying on a *pre-packaged* plan, a naive approach could be to continuously plan the next action to be executed based on the upcoming observations.

However planning is hard; indeed the only classical fragment of the planning is PSPACE-complete [18]; thus some kind of approximation is desirable and even mandatory when the methodology applies for real world agents.

The first improvement is to re-plan only when the plan actually becomes invalid (the cautious agent). Even if the world in which the plan is going to be executed is rather dynamic, only certain facts are actually essential for the validity of the plan. The idea is to keep trace of the action models belonging to the plan in order to obtain the necessary conditions to infer whether a given state reaches the goal through the plan at hand. The first approach in this perspective has been proposed by [40], where the concept of kernel has been introduced.

The construction of the kernel relies on the causal link structure of the plan. A causal link is a particular relation within a plan, which connects two actions A and B whenever A has been inserted in the plan to provide a service for satisfying a precondition of B. That is eff(A) contains an atom q which is involved in the pre(B). The kernel set can be built by following the causal links of the plan in a backward fashion. A kernel will contain not only the precondition of the next action to execute, but also those propositions involved in the causal links starting from previous actions and that still are required for the execution of future actions (even after the next action).

The task does not present decisional point so it can be performed by a simple polynomial function[2].

In Section 2.2 we will see how the kernel can be employed in the context of repair, too.

As a recent work and example of continual planning system, [15] introduces an agent based system for dealing with incomplete information.

[1]When the agent operates under partial observability, also some form of diagnosis is requested; for details see [29],[87],[62],[54],[55]

[2]In Chapter 8 we will show how this kernel can be computed, even in presence of continuous and consumable resources

The approach proposes an agent which controls and executes a partially specified plan of actions. Beyond the traditional actions, such a plan involves also assertion actions. More precisely, assertion actions are specific rules added to the action model. Their contribution is twofold: on one hand they allow the planning phase by asserting facts not known in advance, and on the other hand they allow the execution phase to switch in planning modality when such important information (described in their preconditions) become available. The switching back to the planning phase allows to substitute such action with a concrete piece of the plan.

The absence of information is dealt explicitly in the model but instead of constructing conditional or conformant plans, the agent assumes that such information will be available at some time. So it plans with a limited amount of knowledge and re-plans once this necessary pieces of information becomes available.

2.2 Robust Plan Execution

In this Section we report the state of the art in the field of the robust execution of plans. We firstly introduce what a plan actually is; afterwards, we describe some relevant works that have been proposed to address the issues arising from the plan execution.

2.2.1 Introduction

Traditionally in the Artificial Intelligence community (see [89]), a plan is a (partially) ordered set of actions, given as result of a decision making activity performed by a planner (which may be a human or an artificial agent or a mix of both). In the classical (and generative[3]) setting, given a planning problem P that defines an initial status and a goal, a solution for P is a plan leading the initial state into a state where a set of goal conditions is satisfied. The planner can take actions from a universe of available actions (the domain). There, each action is defined by a set of preconditions and a set of effects. The former asserts under which conditions the action can be considered applicable in a given state; whereas the latter defines the transition in the state once the action is executed.

To make feasible the planning phase (in the classical setting), a given a number of assumptions is done, which typically impose the actions to be atomic, the agent to be the only actor in the world and the action to have a deterministic set of effects.

For example, a classical planner does not take into account that during the plan execution the agent could be in the position of facing undesired action

[3]In opposition to the generative planning there are actually planning based on hierarchical task network (HTN) representation. For an example of that see [74]

effects (she takes the wrong road) or exogenous events (the road may be actually interrupted). Such contingencies may indeed threat the plan consistency. For this reason, to make the execution robust in facing such situations, two strategies are possible:

1. Anticipate the multiplicity of contingencies at **planning time** by equipping the agent with an explicit representation of the uncertainty within both the model of the actions and the initial state (*Proactive behavior :: Off-line reasoning*).

2. React and Replan at **execution time** when necessary by allowing the agent to observe and monitor the environment (*Reactive behavior :: On-line reasoning*).

While the former is applicable when the number of contingencies to consider is limited and enumerable (*bounded undeterminacy*), the latter can be exploited also in context where the agent has no chance to anticipate all the possible contingencies, as it has no sufficient prior knowledge (*unbounded undeterminacy*).

It is thus evident that in domains where both approaches can be applied there is an intrinsic trade-off between the two strategies. Indeed, under the assumption that one can effectively have a reliable and consistent prediction, an agent employing the former strategy will exhibit a very conservative behavior that in some situation may bring it to not find any possible course of actions. On the other hand, an agent ignoring potential failure situations from the very beginning could be trapped in dead-end situations which could arise when the unexpected situation is actually unrecoverable.

For this reason, there is no best solution and the trade-off often depends both on the domain and on the problem at hand. In the next two subsections we will review both points of view.

2.2.2 Off-line: Generating Robust Plans

In this Section we analyze how an agent can generate robust plans at planning time.

In the off-line phase the aim of the planner is to find robust plans by releasing part of the assumptions that typically are made in a classical setting. More precisely the assumptions refers to the deterministic action effects and complete observability of the world. The absence of such assumptions is supposed to guarantee that the execution is intrinsically tolerant to the several *situations* that can be encountered all along the course of actions.

We can distinguish two types of approach, which deal with uncertainty at planning time. The first deals with uncertainty in the more explicit sense; firstly the world state is represented as a set of possible worlds and secondly action effects are allowed to appear in a disjunctive way.

The second approach handles uncertainty in a probabilistic way. Here both the state and the action effects are expressed by particular probability distributions over the domain of the variables involved.

As far as it is concerned by the first family of approaches a plan is searched through the belief states, since at each step of the execution the agent could be situated in several possible world states. The resulting plan is a **conformant plan** in case the agent is not allowed to perceive the surrounding environment[4], otherwise, the resulting solution is a **contingent plan** where the behavior of the agent branches according to the output of a given set of sensing activities.

(Non) Deterministic Conformant Planning

A conformant planning problem differs from the classical one in that the initial state could not be completely known, and the actions (in the non deterministic setting) may have non deterministic effects.

In the classical planning paradigm the agent has a non ambiguous representation of the state of the world. In the propositional setting this is indeed achieved by expressing the state as a set of *true* propositions, assuming *false* the non mentioned ones (Closed World Assumption). Conversely in the conformant setting the state does not have a unique interpretation rather it is represented by a logical formula whose interpretation may entail many possible worlds. For example given two locations, namely `A0` and `A1` one can express ambiguity on the current status of the robot by (explicitly) asserting the formula:

```
(OR (AND (AT ROBOT A0) ( NOT (AT ROBOT A1)))
(AND (AT ROBOT A1) ( NOT (AT ROBOT A0)))).
```

In particular the formula captures that `ROBOT` could be either in `A0` (but not in `A1`) or in `A1` (but not in `A0`).

A conformant planning problem can be solved by forward search algorithms in the belief states space ([13],[51]), through proper heuristic functions extracted by analyzing modified version of the planning graph[5]. The characterization of the belief state space is crucial for the performance of the search to be carried out. To this end, there are two main approaches. The first encodes the belief in on Ordered Binary Decision Diagram (OBDD) while the other is based on SAT[6]. The OBDD allows a compact and at same time explicit representation of the state. Differently, a SAT based representation requires a non trivial computational step for the assessment of propositions inside the belief representation. In a nutshell, SAT based representation trades space for time by forcing the

[4]Due to lack of sensors, conformant planning is also called *sensor-less planning*

[5]The planning graph is a structure introduced with the *graphplan* planning system ([1]) in the context of classical planning. We will introduce the structure in the next Section, i.e. in the context of replanning, once the description will be strictly necessary for the comprehension of the approach

[6]The formulas are represented in a Conjunctive Normal Form (CNF) and are handled by a general purpose SAT solver, as for instance the one reported in [67]

planner to verify on the fly the satisfaction of the action preconditions against the intersection of the possible worlds encoded in the state.

In the last few years also translation based techniques have been presented. For details see [78].

The conformant planning assures that an agent will execute and will reach the goal, despite uncertainty in the action effects and the initial status. However, the approach usually yields too conservative behaviors. Let us imagine a task in which an agent has to reach its work location. The weather is not known in advance and the agent is not allowed to go outside if it is raining. A conformant plan for this problem then requires that the agent has to take the umbrella in every case; of course the umbrella is odd in case the weather is actually good. One could simply provide the agent with the capability of sensing the environment surrounding her to produce a plan for each contingency of the problem. This is what is done in the contingent planning.

(Non) Deterministic Contingent Planning

In the model of contingent planning, the agent in charge of executing the plan has available a set of actions which allows her to discern among (some) characteristics of the world state. Such actions are usually called *sensing actions* and as traditional actions they are equipped with a set of preconditions but, differently from the traditional effects, they provide information on a particular state of a subset of the propositions characterizing the world.

As long as sensing actions are applied the set of possible world interpretations shrinks making the search simpler.

However, when the contingent planning setting allows the presence of non-deterministic action effects, the application of these actions during the search injects new uncertainty that must be newly handled within the belief state.

As for the conformant case, a contingent planning problem can be solved by searching in the belief states. One of the main approaches consists in converting the problem in an AND/OR tree, where AND nodes represents actions and OR the belief status. Given a contingent planning problem, a solution is a sub-tree of AND/OR where each path starting from the root ends in a leaf where the set of goal conditions is satisfied. Moreover, analogously to the technique adopted for the conformant setting, the choice of the belief representation plays a key role in the performance of the system. For further details see [52].

Probabilistic Planning

Another way to achieve robustness is to produce plans that considers uncertainty in form of probability distributions. In this context, the probability can be expressed both in the initial state and as outcome of the action execution.

The model of probabilistic planning has been initially introduced from [57]. Here the problem is defined in terms of:

- an initial ambiguous status where each variable value is associated with a probability distribution;
- a goal which defines a desired configuration of facts to reach at the end of the execution;
- a set of (probabilistic) actions;
- a *threshold* value ranging from 0 to 1 which defines a desired probability degree for the successful plan execution.

A plan for this problem is a sequence of actions that, starting from the initial belief state, takes the agent into a state where the goals are satisfied with a probability equal or greater than the given *threshold*.

In solving a probabilistic planning problem, various techniques based on a fixed length planning problem transformation have been applied; here the representation typically relies on the relative SAT and/or CSP solver extended to deal with probabilities. As for the previous approaches, one of the key aspects is the belief representation. However, while for the logical case one can immediately imagine an OBDD representation, in the context of probabilities models as SAT combined with Bayesian Network seems more appropriate.

Recently, the model of probabilistic planning has been extended and has been used for handling resources and time explicitly. That is, the state is no more represented just in a qualitative manner, but also in terms of a set of continuous numeric variable modeling resources as energy, time and so forth. For details see [10].

Towards Probabilistic and Contingent Planning

The necessity of dealing with continuous variables (representing for instance resources) has been clearly noticed since the adoption of *numeric fluents* in PDDL 2.1 ([42]). However the most of the work done refers to the deterministic setting. An exception is the work reported in [26] where a proposal to combine the model of probabilistic planning with the contingent one has been introduced.

The key idea is to enrich a starting valid solution with additional branches that during the execution may be activated once a set of conditions becomes hold.

For instance let us imagine a scenario where a mobile robot has the task of visiting a certain number M of interesting sites. Let us further assume that the quality of the task increases as the number of visited sites at the end of the mission grows up. Given uncertainty on the value of the energy assumed,

one can imagine that the plan computed only achieves N<M sites because the energy may be not sufficient to visit all the M sites. However, if for some drive action the actual consumption is less than expected, it may be the case that the mobile robot can actually achieve the mission with a greater reward, namely by reaching a higher number of sites of interest. It is of course quite natural to imagine to add further branches to the starting solution in order to encompass also other sites to be visited once some conditions occur (e.g. the power has 10 unit instead of 6).

Of course this kind of approach is slightly different than the ones given in this section. Indeed, the problem addressed by the authors requires that the resulting solution handles both *hard* and *soft* constraints. For further details on over-subscription planning see [92].

In the next Section we discuss how to manage the robustness of the plan on-line, that is, once exceptions and deviations from a set of nominal (assumed) conditions are actually intercepted. This approach overcomes the limitation of the previous solutions as it does not require the prediction capabilities foreseen in a pure off-line approach.

2.2.3 On-line: Robust Execution via Replanning

In this Section we describe how the continual planning agent reported previously repairs its course of actions when the unexpected situation has been actually recognized. In particular, as noticed in [70], each advance obtained in the context of off-line plan generation corresponded in literature to a new method for the repair process, which indeed is typically based on a plan-adaptation step and hence on re-using the effort computed at planning time. Despite theoretical results, [75], it has been showed that adapting the solution rather than computing a new solution completely from scratch is indeed more convenient in practice.

The plan adaptation has its root in the case based reasoning ([3]), which studies how to obtain a solution for the current (unsolved) problem from a set of past (solved) problems. The aim is to exploit as much past experience as possible to facilitate the resolution of the current task. The hope is that solving the current problem from scratch is more difficult than adapting an old solution for the new problem. A survey and an analysis of different dimensions of the plan adaptation problem is reported in [70].

In the next we will give an overview on the most recent works on repair by assuming that such a repair is actually involved online, when the continual planner intercepted that the plan is not valid anymore.

Repair via Planning Graph

A quite recent approach presented in the context of the plan repair is the work proposed in [45]. The main and novel idea is to combine the old plan structure with the *graphplan* representation of the new problem.

The *graphplan* is a very powerful structure introduced by [1] for the purpose of making efficient the propositional planning. The structure represents a given planning problem into a directed layered graph. More precisely, the resulting graph consists of a set of levels where each level encompasses a set of actions and a set of facts. The facts describe what could be true in that level, whereas the actions describe the transition operations that could be applied in such level (i.e. the action whose precondition are satisfied in that level). Each level l is linked to the next level $l+1$ due to the union of the positive effects of the actions in level l and the facts that are true in l. The first level represents the initial status whereas the last is a status in which the goals are satisfied.

For the plan generation problem, such a structure is incrementally built for finding a (potentially parallel) course of actions through the levels. The number of the levels are indeed equal to the time steps of the plan and actions are selected by taking care of the conflict relations, arising by the mixing of the actions and the facts for each level (mutexes, interferences).

In the repair context, and more precisely in the work of Gerevini et al. the *graphplan* structure G is built for the new problem Π_{new}[7] but instead of searching for a plan (as usually has been done in the planning context) the structure is exploited for reasoning on the plan π_{old} provided for the previous problem Π_{old}. In such a perspective only a set of conflicts are considered.

More precisely, the algorithm developed, namely *ADJ-PLAN*, focuses its attention on the inconsistencies in analyzing π_{old} w.r.t. G. As a first step, the algorithm removes the actions in π_{old} that do not belong to G (such actions are considered useless) and for each inconsistency found in π_{old} (an action whose preconditions are not satisfied in that level) establishes a repairing window.

Having established which are the portion of the plan to be repaired, it considers each window as a separate sub planning problem, respectively. The window has a left and a right bound that correspond to an initial status and a goal status of the new sub-planning problem. These bounds are determined by the level in which the action causing the inconsistency appears. That is, given k the level of the action presenting the inconsistency, the initial status is extrapolated by the level $k-1$ of G; whereas the goal is asserted by combining the information of the actions of level k and some additional facts (computed heuristically for speeding up the process, see below). As long as the solution is not found in such a window, the algorithm moves the right bound of the window ahead in G.

[7] We will use the Π symbol to identify planning problems; whereas we will use the lowercase version for its solution, i.e. a plan π

As anticipated, the approach builds the right bound of the window heuristically, using two main methods based respectively on a persistence assumption and on the causal link structure of the plan. The ratio is that the replanning window goal set should also include those effects which may be exploited as a support of future actions in the plan. Conversely indeed it may be the case that the replanning window will propagate an inconsistency for the rest of the plan, meaning that new repairing windows could appear. Of course finding an exact estimation could be difficult as much as replanning completely from scratch. This is the reason why an heuristic estimation is preferred. Despite this approximation, the process does not affect the completeness of the approach, in the worst case a bad heuristic may degrade the efficiency but it will never compromise the space of the solutions.

It is worth noting that such an approach can in principle degenerate into a replanning from scratch when the length of the window becomes the length of the whole plan of actions. In such a case the approach will perform worse than replanning from scratch due to the overhead for building the replanning windows.

One of the shortcomings of the approach refers to the building of the planning graph. As a matter of fact, planning graph works on factored action representations instead of schemata representation (e.g. PDDL actions). For this reason, when dealing with action schema, the planning graph requires a (potentially expensive) phase for grounding the actions at hand, which causes a non negligible blow-up of the planning graph structure. However this is not only a repair problem but also a problem for the first planning phase, thus the proposed solution still represents an elegant approach for a plan computation, being also capable to deal with problems that differ in their set of goals.

An alternative way of looking into the replanning topic is related to the refinement planning.

Repair via Refinement Planning

Another approach investigated in the context of the classical planning is the refinement planning, which has been introduced in [56]. In this case the search for a valid plan is computed by trying to improve a starting (flawed) solution until a valid one is found. Therefore, it is evident that the approach is quite appropriate in the context of repair.

Given the planning problem, the strategy starts immediately by formulating a first (flawed) solution which is composed by just two pseudo actions representing the initial state and the goals. The first one has empty preconditions and effects equal to the initial state, while the second one has empty effects and preconditions equal to the goal statement. A refinement planner iteratively adds actions until a plan with no inconsistency (i.e., there are no action with

open preconditions) is found. Each action must solve at least a part of the open precondition. When no action can be applied, but the solution is still consistent, a backtracking is invoked.

In the work [97], the authors propose to see the repair as a direct extension of the refinement strategy, in which the actual process for the actions insertion is anticipated by a further step, namely an *unrefinement* step. The key idea is that an unexpected contingency invalidating the plan may actually require to delete actions rather than adding new ones.

Similarly to the refinement phase, the unrefinement step generates a new set of candidate plans. Indeed for each plan a number of actions can be deleted. Of course an exhaustive search would cause a combinatorial explosion of cases to consider, thus the unrefinement relies on two filter steps. The first is a mechanism that privileges the deletion of actions which are in the start and in the end of the old plan; whereas the second chooses the action to delete according to an heuristic estimation of the distance between the resulting candidate plan and the valid solution. The heuristic measures the effort for solving the inconsistencies caused by the open preconditions of the new candidate plan.

Once the actions to delete have been selected the algorithm starts its refinement process till a valid plan is not found. From this point to the end of the process, the unrefinement step is not repeated again.

The heuristic function employed in the process is inherited by the recent advances on heuristics based on the distance among state, which have been successfully employed in the context of partial order planning (POP) (the field where the refinement planning takes inspirations). The first work in combining the search for Partial Order Plan and this kind of heuristic functions is reported in [103].

The work turns out to be a clever general template for repairing plan. One of the drawback here is the lack of a management for an explicit online framework. As a matter of fact, the plan execution phase has not received the proper attention as well as there is a lack in handling time bounds within which the repair has to be performed. The question indeed is, if there is no bound on the amount of time to be spent by the replanner, is it really so important to repair instead of replanning from scratch?

Repair via Regressed Goal States

Planning via heuristic search in the state space is probably the most successful approach to the (classical) automated planning. In the last decade we assisted to an intensive use of such a paradigm in the planning community, probably due to the facility in combining the search through the state space with the

use of the delete relaxation heuristic[8] ([14]). This kind of operation has been become a central part of many planners (with some differences on how these were developed) and it is one of the main reasons for the success of the winners in the recent planning competitions.

The idea in the work of Garrido et al. ([44]) is to combine the distance based heuristic with the continuous planning agent (see [89]). The intuition relies on an important consideration of what may typically happen during the action execution. More precisely, given a plan of actions, one can imagine the execution of such a plan as a sequence of states, where the starting state is the initial state and the final state satisfies the goal conditions. One of the reasons which may cause the invalidation of the plan is that, because of unpredictable exogenous events and/or undesired effects, the actual observed state may do not match (or better may be not compatible) with the expected state. Substantially, there is a deviation of the starting trajectory such that the remaining part of the plan is not able to take the agent in a status in which the goal is satisfied.

In this perspective one can effectively think of the repairing process as the process of searching for the closer point of the trajectory. While the continuous planning agent does not specify how actually estimate such a distance, it is possible, by exploiting the distance among states achieved by the heuristic described above, to define in an effective way a strategy in measuring such distances.

The Garrido approach, starting from this idea, developed an anytime algorithm that incrementally searches for the best bridge to build from the current (unexpected) state and the plan that has to be performed. As a first step the algorithm rebuilds the trajectory of states to be covered by identifying only those set of atoms which are effectively needed for the plan at hand. Since the approach also support durative actions, the concept is a light extension of the kernels developed by [40]. Each kernel is here called subgoal. Whenever an exception is identified, that is a situation that does not support the current subgoal, the online monitoring triggers the decision module that has to promptly responds whether it is the case of repairing or replanning from scratch. The reasoning here is performed by estimating the distance to the subgoal and the final goal. This estimates the effort of repair against replanning.

Whether the decisional module selects the repair, the algorithm generates a new planning problem whose solution (if any) will be combined to the remaining part of the plan. Moreover, the approach will continue to analyze the next subgoals to understand if there is a better bridge (in terms of cost of the plan). The algorithm terminates when either the search is exhaustively performed (also the final goal is tested) or a time threshold is reached.

The advantage of this approach is an effective strategy for an on-line context.

[8]The delete relaxation heuristic consists in estimating the distance by two states by solving the relaxed planning task in which actions are considered only in their positive effects

Of course the way in which the first decision is taken may be to risky. Indeed it is possible that the first subgoal would be actually unreachable whereas the subgoal immediately after it would be (and even with very few actions). This careless behavior may in fact produce an immediate chose for replanning from scratch.

As argued by the same authors, the approach typically yields stable plans since the agent tends to come back to the choice made at planning time without completely unsettle its intentions. The stability is a quite relevant property, as we will see in the next paragraph.

Repair to Achieve Stability

As multi-agent domains have been more and more investigated in the field of automated planning, also the repair context has received an increasing amount of interest.

In these domains, especially those where each agent has the possibility of applying local repairing techniques in an independent way, the impact of the decision taken for facing unexpected situations may become a critical issue. Indeed, as long as decisions diverge from the choices taken at planning time, the behavior of the agent turns out to be very difficult to predict from an external observer, if either she is a human or it is an artificial agent. Moreover, it could be the case that the agents involved in the process have provided expectations and agreements on the choice at planning time. For instance the preconditions of other agents may depend on the actions of other agent's plans.

Therefore the repair has to preserve as much as possible the old structure for the purpose of minimizing the impact on the rest of the system (intended as other agents). In few words the plan has to be as much stable as possible. As studied in [101], the repair mechanism could cause a rapid combinatorial explosion of cases to consider.

The stability requirement can be imposed from the very beginning by identifying in the model of the actions those mandatory constraints for an effective agent cooperation. This could be achieved by specifying further constraints to be taken into account during the repair process. Another option would be to agree for a safe state to be reached in case the mission is not achievable anymore. This idea is reported in [63].

A different solution proposed in literature is to measure quantitatively the similarity between the plans. In particular, [60] introduced a new stability metric, defined as a function of the distance between two plans. That is, assuming to have a distance between the plan A and the plan B, we will say that as the D(A,B) increases, the Stab(A,B) decreases and vice versa.

Concretely, given two plans π and π', Fox's stability measure is achieved by summing the number of actions belonging to π that do not belong to π', with

the number of actions in π' not belonging to π. As stated by the same authors, the introduced metric is a preliminary definition. As a matter of fact, the metric suffers of some limitations: firstly it does not take into account the order of the actions, and secondly it is not aware of similar actions.

For the purpose of producing repair mechanisms being able to achieve stable plans, the work of [44] proposes to account of the stability when deciding between repair and replanning. As a limitation however the approach does not account the stability when the anytime search starts.

Instead, the work of Fox et al. ([60]) explains how to steer the anytime behavior of the Lpg-Td planning system ([45]) to (heuristically) privilege only stable solutions.

2.3 Robust Schedule Execution

In this Section we consider a plan of activities given in form of a schedule, that is, the agent is now more interested in understanding precisely when actions have to be executed; here each action is indeed represented by a pair of events representing respectively its *start* and *end* time. In addition to the causal aspect managed by traditional planning system, the temporal dimension of a plan is another valuable characteristic to be considered in the context of planning and execution.

After a brief introduction which will recall the background notions in the field of scheduling plan of actions, we will report two recent works that addressed the problem of robust execution of a schedule.

2.3.1 Introduction

In Section 2.2, we analyze plans of actions where each action describes, in a logical way, its applicability conditions on the world and the effects caused by its execution. The main limit within this model is that it has a very action centric nature; the modeler has no chance of expressing relations involving set of actions. In particular, the classical model does not allow to express (potentially complex) temporal constraints among actions as well as a way for stating quantitatively when these have to be executed (e.g., action a has to start at seven o clock) and what their duration is. Instead, the classical setting sees the order of the action as an implicit consequence of the plan generation. That is, the order is the actual output of the planning process.

The STP Model. A classical scheduling problem focuses its attention on the temporal dimension. Initially introduced by Allen et al. in [6], scheduling actions problems are in the recent years seen as an instance of quantitative temporal reasoning tasks, and are generally handled by Simple Temporal Problems

([33]) or its extensions (for instance see [94]). The Simple Temporal Problem (STP) is a special kind of a Constraint Satisfaction Problem where:

- Variables represent temporal events; the domain of a variable is R^+
- Constraints encode temporal *distance* among events; each constraint is represented by a binary inequality relation, e.g., given two events X_0 and X_1 we may say that $X_1 - X_2 \leq r_0$ where r_0 is a real constant

Similarly to a Constraint Satisfaction Problem, an STP can have a graph formulation. Precisely an STP can be formulated as a weighted directed graph where nodes are the variables involved, arcs express the order relation among the events, while the weight of such arcs represents their temporal distance.

The consistency checking task for a STP is efficiently solved by verifying the absence of negative cycles within the graph network associated to the STP ([33]). Whereas making explicit all the relations and the minimum temporal distance on the network can be performed by the well known Floyd Warshall's algorithm in $O(n^3)$. Recently, the *All-Pairs Shortest Paths* problem arising from the STP has been attacked in [80].

From an ontological point of view, the temporal reasoning behind the scheduling considers time as an unbounded, dense and linear set of points.

Encoding Actions in STP. A scheduling task starts by encoding the set of actions and their relations as an STP. In particular, each action is subdivided into two events: the former defines the *action start time* and the second the *action end time*. By solving the associated STP, the scheduling task generates a set of pairs for each action which specifies the domain of its (the action) *start time* and *end time*. Such domains, by definition of the STP, are consistent w.r.t. the constraint imposed on the network.

It is worth noting that, while the classical planning is interested to find such an order, here the ordering is actually a part of the problem specification. The integration of the both aspects in a unified vision makes the problem very hard ([84]).

After this little preamble which introduces ourselves in the scheduling context, we can roll back to the main objective of this section; that is to explain how to assure that the execution of a given scheduling of actions can be made robust to unexpected (temporal) events. Intuitively, the objective of the agent in handling schedule of actions is in preserving the temporal relation given a certain network of constraints among actions of the plan. That is, the underlying STP is required to be consistent throughout the plan execution.

In this field the STP can be checked and revised by seeing the same STP as a dynamic structure ([24],[22]). The real issue nevertheless arises in real world scenarios where the agent is required to deal with limited amount of resources. For this reason, the next Section shows an approach to robust schedule which

considers both temporal and resource constraints. Afterwards, we observe how an STP model can be extended to support flexible execution of plans with choice.

2.3.2 Combining Simple Temporal Problem and Resource Problems

In real world applications, the plan execution often relies on a given number of resources which in turn are typically *finite*; therefore, given a temporal interval, only a limited number of actions needing such resources can be executed at the same time. The user modeling the task is supposed to insert temporal constraints (for instance in form of Simple Temporal Problem) in order to assure a correct use of these resources. However, it is desirable that some constraints can be intercepted autonomously by giving a declarative representation of the problem.

In the scheduling community, combining resources and temporal constrained tasks in a unified point of view has been defined as a variant of the job shop scheduling, namely RPCSP-max.

An RPCSP-max is the problem of finding the starting time for a set of activities given that:

- each action a_i has a duration dur_{a_i}
- each action a_i requires a given amount of resource r_k till the end of the execution
- resources are limited, i.e. each resource r_k has an integer capacity $max_{r_k} \geq 1$
- there are further ordering temporal constraints among actions in the form of $c_{i,j}^{min} \leq s_{a_i} - s_{a_j} \leq c_{i,j}^{max}$

A solution for the RPCSP-max is optimal whenever the resulting makespan is minimum, which is the total amount of the time spent to accomplish all the activities involved in the task.

In the work of Cesta et al. ([82]) the problem has been reformulated in the context of robust plan execution. More precisely, the authors defined the concept of Partial Order Schedule. A Partial Order Schedule is a collection of actions which are consistent with respect to both temporal and resource constraints. The resolution of the RPCPS-max is done by reducing the problem in an STP, enriched with further constraints given the requirements of actions on the resources.

Instead of focusing on a particular allocation of the action in the time, the POS preserves the STP starting structure also for the execution phase. The main idea is to leave the agent executor as much freedom as possible in choosing

the final ordering of the actions. For this reason, the agent has to be aware of only the precedence constraints, which are mandatory for the consistency of the starting temporal requirements and the resource at disposal.

The robustness here is intended in that schedules have to be as flexible as possible. The flexibility in turns depends on the structure of the obtained POS, and in particular a POS flexibility increases as the number of constraints among actions decreases.

The process takes in input an initial STP and a number of limited resources accompanied with a description asserting the relation among actions and resources. The output is an STP, namely a POS which is supposed to be consistent w.r.t. the resources and the initial constraints.

To compute such a solution, the work proposes two main strategies. Both share a general greedy algorithm while differ in the way in which the resource profile is computed.

As a first step the general greedy algorithm is invoked with the starting STP; then the STP model is checked against the resource constraints. Whether the checking reveals an inconsistent STP at least a precedence constraint has to be imposed within the STP. If for instance two actions compete for the same resource, one of them has to precede the other one. The approach heuristically selects such a constraint by looking into the whole set of pair of activities which conflict with each other.

After the pair selection, the algorithm is recursively called with a modified STP model (the one with the new precedence constraint) until a consistent solution (in the STP and resource sense) is detected.

As anticipated before, the approach can follow two different strategies. The first strategy is based on the concept of resource envelope proposed by [72]; it proceeds by refining all the possible schedules until only the set of valid ones is found. The resource envelope is built by considering all the maximum and minimum resource usage within the POS at hand.

The second strategy takes the problem from the opposite direction. It starts from a well defined solution (the one in which actions are considered to start at the earliest valid time), and compute the resource profile for that specific solution.

Note that while the former strategy finds Partial Order Schedule by definition, the second strategy does not. For this reason, the second strategy is followed by a post-processing step which substitutes all the computed constraints with the necessary ones given that particular assignment (the earliest start time). The idea is to extend a particular solution within a new POS to include a greater set of solution which are still consistent with the starting formulation of the problem. The strategy inherits the chaining procedure reported in [21]. Recently, the problem has been attacked by [8], where a complete

resolution approach has been presented.

Speaking of robust execution, this is one of the most effective approach in trading off expressiveness and efficiency. Indeed, the Simple Temporal Problem represents a big class of tractable problems and there are many efficient algorithms based on its graph representation.

As a shortcoming, however, it must be noted that such an efficiency is paid in terms of a lack in the chance of adding further actions. That is, as well as flexibility is guaranteed from a temporal perspective, there is no flexibility in terms of different course of actions. The STN model does not capture different set of events, and hence of actions, but only different ordering of such events. In many real world scenarios this can be a strong limitation. For this reasons some works have dealt with the problem by extending the STP model.

2.3.3 Towards Powerful Schedule Representation

The STP model has some drawbacks. Indeed it is just a special kind of Temporal Constraint Satisfaction Problem, and its limits have been identified since in the work of [33]. For this reason in the planning and scheduling community more expressive formalisms have been investigated and a number of extensions has been proposed ([99],[94]); among them the Disjunction Temporal Problem model received the most attention in the last years, above all in the context of robust execution.

A DTP extends the STP model by allowing to combine the constraints set by means of disjunctions. For instance in the STP model the user is not allowed to express constraints of the form $X_0 - X_1 \leq r_0 \cup X_3 - X_2 \leq r_0$, because of the language is limited to handle only conjunctive set of constraints.

Albeit the DTP seems just a little extension, the class of the scenarios that the DTP is able to capture is much larger. This is of course paid by a step ahead in the complexity for the worst case; indeed the adoption of a formalism allowing for disjunctively representing constraints makes the temporal reasoning task intractable ([94]).

In the context of robust schedule execution, [27] proposes a model based framework where plans are represented as a reformulation of Disjunctive Temporal Network. To comply with the STP formulation, they called such a model Labeled Simple Temporal Network (LSTN). Similarly to the Disjunctive Temporal Problems, the disjunctive nature of the LSTN allows the agent to select and adapt the execution with the more proper sequence of actions. One of the limitations indeed of the Simple Temporal Problem is due to the fact that the model assumes that the events always happen; for this reason a behavior of a system described by this model can have the flexibility of just changing the order of the events (and hence of the actions). The LSTN includes the possibility of disjunctive set of events. As a matter of fact, two given solutions within the

LSTN may differ in the set of events considered. This increases the flexibility of an agent to perform different set of actions.

The DTP resolution task has to be carefully managed since in a context where efficiency is a major constraint, the DTP resolution cannot be delegated to the online phase. In this perspective, [27], by taking inspiration from the Assumption Truth Management System (ATMS) developed by de Kleer in [30], proposes a system called *DRAKE* which performs a thorough investigation of the problem from the very beginning.

The idea is to anticipate all the action selections that can be performed by the agent during the execution. Each decision indeed has a set of implications on the remaining part of the task to be executed. The selection of a particular event within the DTP may cause the selection of new set of constraints for the allocation of other events.

This can be retrieved by analyzing the way in which a reasoner deals with the consistency checking of the underlying DTP. We can indeed see that a DTP is consistent if at least a disjunct for each disjunction is satisfied.

Intuitively in the context of plan represented as DTP, each action is actually a disjunct of the temporal net. When an action is selected, the DTP propagates the implication of that particular decision throughout the constraints of the graph. The result is that only consistent future disjuncts are preserved. Thus, instead of focusing on a particular assignment of disjunct, the final structure (the LSTN) is supposed to store all the possible trajectories of events which are consistent with the DTP at hand.

The DRAKE aims at achieving a cheap (in the size) dispatchable Labeled-STN which can be used as a scheduled plan of actions with choice. Here dispatchable means that the agent, relying on the labeled STN can select effectively in real time only the actions consistent with the DTP, without performing any kind of reasoning.

One of the shortcomings of the approach is a complete lack in the resources management. However, even if in principle it may be possible to consider all the impacts of the actions selection on the amount of resources, it may be the case that the size of memory for representing these further implications would become a critical issue.

2.4 Conclusions

This Chapter reported the state of the art on the robust execution by breaking the discussion in three main topics; that is, the *architecture for autonomous systems and continual agent paradigm* which looked at the problem from an higher point of view, the *robust plan execution* and the *robust schedule execution*, which instead are focused on the way in which a task can be represented and

managed.

In particular Section 2.1 focused on the three tiered architecture proposed by Alami's and on the continual planning paradigm. The continual planning is supposed to provide a solid domain independent framework for the development of real world agent. However, to the best of our knowledge, most of the challenges of the continual planning are still unsolved and the real-world applications often opt for domain based customized approaches. The most of these works inherit the multi-level approach of the Alami's architecture.

In the latter sections we discussed a number of approaches which differ (i) on the model adopted for representing task, (ii) on the methods applied for making its execution as robust as possible. We noticed that in principle it is possible to build robust solutions since the very beginning, but while it turns out to be feasible for the management of the schedule (e.g. [27]), we have seen that it can be very hard when the task to be managed is a traditional plan. As we have argued in 2.1, an agent to be effective has to close the reasoning and perception loop in an acceptable amount of time. However the complexity results in planning under partial observability imposes very hard barrier for the efficiency of such systems ([18], [85], [84]).

In the context of on-line repair, as also stated by [70], we observed that each progress obtained in the area of automated planning has been followed by a new repair strategy. It is in fact quite evident that the way in which the search is performed in what is called the first-principle planning ([31]) are often extensible for more efficient repair mechanisms. For instance, the work [2] where the *graphplan* structure is deeply employed and the proposal of [97] where the authors extended the refinement planning in the context of repair.

A similar phenomena is observed in the context of the robust schedule execution, where the most recent works have a strong characterization in what the underlying temporal model employed is ([81],[27]).

As well as these works describes concretes methods for performing the repair they often ignores the continual planning framework they are supposed to belong to. In particular the works in the *plan* repair seems most suitable for the off-line phase (in a case based planning style). Differently, the works on the robust schedule execution seems very focused on the problem of the execution even if their flexibility (which is supposed to be the main way to achieve robustness) is mainly handled from a temporal point of view.

As each categorization, the discussion reported makes some approximation. There are contributions in literature crossing these macro areas in different ways; for instance there are approaches dealing with both scheduling and planning where the continual planning (even if not specified) is the underlying spirit. For details see ([88],[95],[71],[73], [20], [19], [93]).

Chapter 3

Robust Plan Execution via Multi Modality Actions

This Chapter introduces the main approach adopted in this thesis by presenting the formalism of reference. In particular the Chapter proposes the notion of Multi Modality Action (MMA). An MMA allows to model actions at two different levels of abstraction. The first one aims at capturing the high level behavior of an action while the lower describes the impact (and the requirement) of the action with respect to a given set of resources modeled as numeric fluents. The MMA represents the basic block of an MMA plan which is in turn key in the mechanisms developed by FLEX-RR (Chapter 6) and ACTS (Chapter 9). This notion is crucial in addressing the robust plan execution in a more efficient way.

3.1 Introduction

In the state of the art we revised different techniques that have been employed in the last years to allow agents to execute task in a robust way. We observed that the first main difference among the investigated approaches is the model adopted for representing the action behavior. In particular we identified two main approaches. The first set of works models the execution by means of traditional plans where actions are expressed in terms of preconditions and effects (robust *plan* execution). The second family of approaches focuses on the temporal relations among actions (robust *schedule* execution).

This thesis fits in the robust *plan* execution field and in particular in the family of approaches which tackles the robust execution via online replanning. The main open problem pursued by this first part of the thesis (till Chapter 7) is to extend the continual planning framework for dealing with continuous and consumable resources, by making attention in keeping plans quite stable.

Effectively, the framework should guarantee:

- an efficient strategy for the repair of the current plan when some unpredictable contingency arises (e.g., the resource consumption is greater than expected). Only via an efficient repair mechanism we can be sure that the decisions are provided in time to be useful.

- a limited impact on the plan when the contingency is handled, meaning that the plan must be preserved as much as possible. Intuitively in fact, the less impact on the plan, the more stability will be achieved.

The objectives of this thesis stem from two main considerations. Firstly, we start from the observation that real-world scenarios include both reusable and consumable resources; thus a plan has not only to achieve a set of goals, but it has also to conform to strict constraints on the amount of resources to be used by the agent. Secondly, we observed that in many domains it is important that the online decisions do not prominently diverge from the commitments and expectations provided at planning time ([28]), e.g. with other agents. In fact, the behavior of the agent can become difficult to predict in case the repair completely unsettles the previous plan.

As we have seen in literature, however, while there are a lot of works in dealing with the problem of plan execution and replanning in the classical setting ([2],[28],[44],[97]), just few works consider resources constrained tasks. In those works, furthermore, resources are mainly modeled as just reusable and the interactions with the causal aspects have been dealt just with reference to the temporal point of view ([82],[8],[27]).

The difficulty to find an efficient way to reuse the old plan is made evident since the theoretical point of view. As stated in [75], approaches based on a plan adaptation step are at least difficult as replanning from scratch (especially in those contexts where the conservatism, i.e. the minimal perturbation planning ([28]) is quite relevant)[1]. Replanning from scratch, in turn, corresponds to create on the fly a new instance of a planning problem whose resolution may be very hard from a computational point of view - the only classical planning is a PSPACE-complete problem, [18]; therefore a more radical approach seems inevitable to achieve some theoretical guarantees.

In this thesis, and especially in this Chapter, we start to approach the problem from a different perspective by reconsidering the issues from the very beginning, i.e. from the model of the actions.

In particular we observed that (similarly to [42]), it is necessary to explicitly model the (expected) profile of resource consumption during the execution of

[1]It is worth noting that the results just apply for the classical setting. The choice to deal with consumable resources imposes further challenges in understanding under which conditions the adaptation could be feasible.

any action. We thus model consumable resources as numeric fluents, which are explicitly mentioned in the preconditions and effects of the action templates[2].

We also observe that, in many real-world domains it is possible to identify actions performing the same task (i.e. obtaining the same effects), but requiring different configurations of the agent. These actions share the same qualitative objectives, while they differ in the way they are performed. Typically, different configurations have different resource profiles. We therefore propose to group these similar actions by introducing the notion of *multi-modality action* (MMA). An MMA represents in a compact form alternative ways, or alternative *execution modalities*, to accomplish a task. In planning terms, we would say that all the execution modalities of a given MMA reach the same set of propositional effects. However, since they are characterized by specific resource consumption profiles, they differ in their numeric preconditions and effects.

As we will see in Chapter 4, the notion of MMAs will allow to see a significant class of repair problems as reconfiguration problems; thus the agent will have an explicit way of controlling just a portion of the action behavior resulting in a (hopefully) saving of computational time and in a limited impact on the structure of the plan. Afterwards Chapter 5 explains how transforming the reconfiguration problem in a standard CSP formulation, which is one of the main innovative blocks for the FLEX-RR architecture presented in Chapter 6.

In the design of the MMA formalism, one of the main criterion adopted is the compatibility with PDDL formalism. This has several advantages. Firstly the MMA formulation can inherit (a part of) the well defined syntax and semantic established in many years of work by the planning community. Secondly, the MMA management can take advantage from the many tools at disposal for reasoning about PDDL actions (e.g. automated planner, knowledge engineering tools and so forth).

The Chapter starts describing formally how we represent a state of the world. Then Section 3.3 introduces and formalizes the notion of Multi Modality Action, while Sections 3.4 and 3.5 report concrete example of MMA action.

3.2 Modeling the world state

As a first point we have to establish, how the state of the system can be characterized in our framework. More precisely, by taking inspiration from the PDDL formalism [42], we agree to a world representation where entities can be represented by a finite set U of physical objects. Moreover, starting from the consideration that objects (e.g. a *site* of interest) are characterized by a number of properties (e.g. the *name*, the number of samples to collect) and may be also

[2]Numeric Fluents have been introduced since the 2.1 version of PDDL (see [42]). PDDL stands for Planning Domain Definition Language, which is the standard de facto formalism for representing planning problems

in relation among each other (e.g. there is a *road* connecting the *site* A to the *site* B), we allow to explicit such relations and properties in two ways.

The first kind of relation is qualitative. For instance we allow to say in our language that, given two objects identifying two sites A and B, the relation *road(A,B)* allows to express the existence of a road between such two locations. The second kind of relation is a quantitative relation stating numeric properties of the objects. For instance we allow to model the power of a robot R by means of the predicate *(= (power R) 10)* as well as we allow to model the distance among the two interesting sites by means of *(=(distance A,B) 100)*.

More formally we can hence define a particular World State Domain as follows:

Definition 1 (World State Domain)**.** *The domain of a world state is a triple* $< U, F, X >$*, where* U *is the set of physical objects,* F *is the set of qualitative (i.e., propositional fluents) relations involving objects in* U*, and* X *is the set of quantitative relations over* U *(i.e., numeric fluents). While the propositional fluents are the traditional STRIPS relation, the numeric fluents are n-ary function symbols mapping n objects of a domain (or variables when used in action schema, see Chapter 4) to a real number.*

It is easy to see that in particular F and X define the domain in which qualitative and quantitative properties can be stated. This means that the domains of F and X consist of all the possible instantiations of relations w.r.t. the objects in U.

Each numeric fluent may also have 0-arity. Such a special case includes the possibility to model quantities that are independent from the physical objects of the domain (e.g. time, costs and so forth).

Given the domain defined above, a particular instance of the qualitative and the quantitative properties of the objects in U represents a world state S of the system; formally we can hence define the world state as:

Definition 2 (World State)**.** *A state* S *is a pair* $< propFluents, numFluents >$*, where:* $propFluents \subseteq F$ *is the subset of propositional fluents that are true in this state, and* $numFluents$ *is an assignment of real values to all the numeric fluents in* X*.*

The fluents in F not mentioned in $propFluents$ for a given state S are considered false (Closed World Assumption) in S. Whereas $numFluents$ is a total function which maps each numeric fluent to a specific real value; for this reason, $|X| = |numFluents|$. Given F and X, it is quite clear that the states space of our system is 2^F x $\Re^{|X|}$.

To simplify the notation, in the following we will denote the propositional and the numeric part of a state S by S_{prop} and S_{num}, respectively. Analogously

to other planning systems, both *propFluents* and *numFluents* could have the special value $undefined$.

In order to comply with several logical based frameworks, as for instance the same PDDL, we refer to relations and predicates over these objects by means of a Polish Prefix Notation. That is, propositional and numeric fluents have the following form:

- propositional : `(prop_relation_name(object1,object2,..objectN))`
- numeric: `(= (num_relation_name(object1,object2,..objectN)) r)`
- where:
  - the i-th object is a concrete object in U
  - *prop_ relation_ name* and *num_ relation_ name* are symbols/alias for the name of the relations/properties
  - r is a number in $\Re$

In particular, apart from some exceptions we will indicate, the rest of the thesis agrees to the PDDL 2.1 grammar defined in [42].

An example of initial state S involving a planetary rover (i.e. *r1*)[3] and a number of sites of interest (i.e. *A,B,C,D*) is reported in Figure 3.1.

```
/*qualitative information*/
(in (r1,A))
(road (A,B))
(road (B,C))
(road (C,D))
(road (A,D))
/*numeric/quantitative information*/
(= (distance(A,B)) 15)
(= (distance(B,C)) 35)
(= (distance(C,D)) 45)
(= (distance(A,D)) 10)
(= (power r1) 1000)
(= (memory r1) 4)
(= (time) 0)
```

Figure 3.1: Initial Status for the Planetary Rover

By observing the Figure, let us note that, while the propositional fluents in S_{prop} give information about the relations existing between a domain object with other objects, the numeric fluents in S_{num} refine the description of the objects with some further properties. In particular, in our thesis we will see that

[3]The domain of the *Planetary Rover* will be introduced in Section 3.4

numeric fluents are the tools for modeling the amounts of objects' resources by means of real numbers.

3.3 The Multi-Modality Action Model

Taking into account the notion of system state introduced above, we now discuss how the system state evolves as an effect of the action application; in particular, this Section introduces the Multi-Modality Action model (MMA).

As anticipated in the introduction, one of the contribution of this thesis is the notion of execution modalities. Execution modalities are intended to point out the different ways in which the *same* action can be performed, e.g., using different devices and/or parameter configurations. The intuition is that the expected results of an action can actually be obtained in many different ways by setting *how* the action is configured. Obviously, different configurations have, in general, various resources and time profiles. Therefore, it may be possible that the execution of an action in a given configuration leads to failure, whereas the same action performed in another configuration does not.

The MMA model extends the PDDL action model by allowing to express, within a single action, the different alternative modalities in which that action can be performed. To keep the explanation simple we consider grounded action instead of action template. The extension towards the *schema* is straightforward.

Formally, we define an MMA as follows.

Definition 3 (Multi-Modality Action)**.** *Given the problem domain* $< U, F, X >$*, a Multi-Modality Action (MMA)* a *is the tuple* $< propPre, propEff, mods >$ *where:*

- $propPre$ *is a conjunction of positive propositions defined over* F*; it describes the applicability conditions, in propositional terms, for* a*.*
- $propEff$ *is a conjunction of propositions defined over* F*, expressing the effect of the application of* a*. This set is partitioned into positive and negative propositions, typically called the add list and delete list of* a*, respectively.*
- *mods is a collection of modalities. Each modality* m *defines a specific way of performing* a *and is modeled as a pair* $< numPre, numEff >$ *where:*
  - $numPre$*: is a conjunction of conditions that must be satisfied for applying action* a *to a state* S *with modality* m*. Each condition in* $numPre$ *is a triple* $< exp1, comp, exp2 >$ *where* $exp1$ *and* $exp2$ *are linear expressions defined over real constants and over* X*, and* $comp \in \{<, \leq, =, \geq, >\}$

– *$numEff$: is a conjunction of assigners which specifies how numeric variables change by applying action a in modality m. Each assigner is a triple $< op, f, exp >$ where $op \in \{increase, decrease, assign\}$, $f \in X$ and exp is a linear expression involving real numbers and fluents in X.*

As many other representation formalisms handling numeric expressions, also in our framework we limit ourselves to linear expressions (see for instance [50] and [45]). Thus, the numeric expressions we deal with are recursively defined as:

- a real number in $\Re$ is an expression;
- a numeric fluent in X is an expression;
- given two expressions $exp1$ and $exp2$, then also $\{exp1\ op'\ exp2\}$ is an expression, where op' belongs to {+,*,/,-}.

The limitation to linear numeric expression concerns the way in which expressions are constrainted to be combined among each other. More formally an expression as defined above is linear iff for each $\{exp1\ op'\ exp2\}$, if op' is either $*$ or / then one between $exp1$ and $exp2$ has to be a constant, i.e. a real number.

The limitation to just linear expressions is due to two main reasons. Firstly, this form of expression allows us to generate normalized constraints in the CSP encoding (see Chapter 5)[4]. Secondly, we comply with the assumption of the most of the numeric planning system ([50],[45],[46]). In fact, as deeply studied in [50], the restriction proposed permits the operation involved in the action effects to be monotonic. Only under this kind of assumption, even computationally still hard, the planning with numeric fluents seems feasible. The monotonic property is one of the block to achieve decidability in the numeric planning setting (for details see [47]).

Observing the model of the MMA introduced above it is quite interesting to note that the action model involves two levels of representation. The former, more abstract, represents the high level preconditions and effects. Whereas, the latter refines the action behavior by modeling how the selection of a specific modality causes variations in numeric/metric terms in the state of the system, for example for consumable resources.

We generated the *abstracted* action of an MMA as follows:

Definition 4 (Action Abstraction)**.** *Let a to be an MMA, its abstracted action is an MMA b with the same set of propositional preconditions and effects of a, but with just one modality at disposal whose numeric preconditions and effects are set to be empty.*

[4]It is worth noting that expressions will be evaluated after an invariant analysis. For this reason, actually, some of non linear expressions can be simplified to be linear, see Chapter 5 for further details.

The abstracted version of an MMA just defines the qualitative high level behavior. It is worth noting that the abstraction of an MMA action is a STRIPS action.

The MMA can be also seen as multiple PDDL actions. Figure 3.2 reports the relation among an MMA and a traditional PDDL 2.1 action.

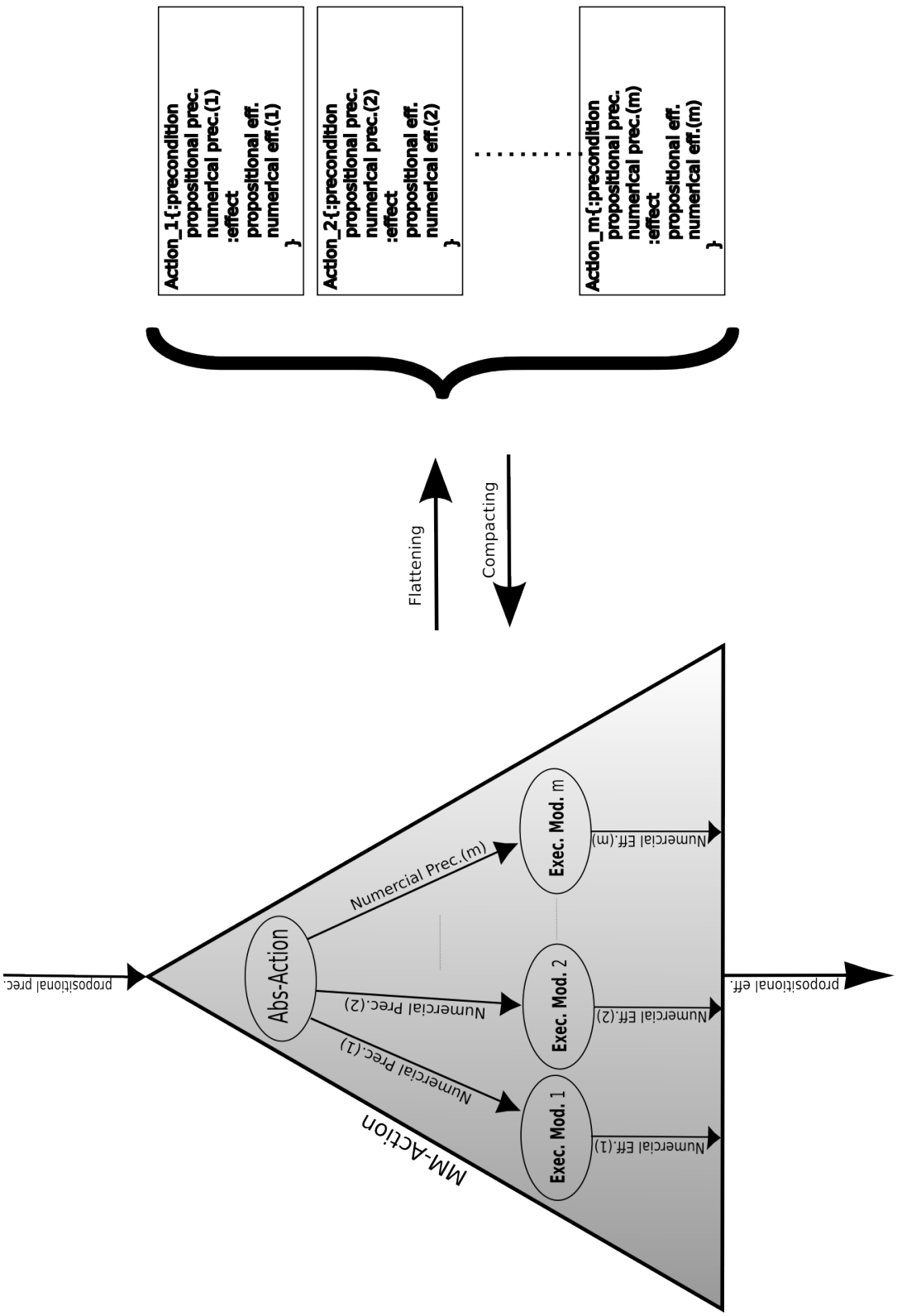


Figure 3.2: From an MMA to a set of Traditional PDDL 2.1 Actions and vice versa

Depending on the necessity, one can convert an MMA in a set of PDDL actions (flattening) and group a set of PDDL actions in a MMA as well (compacting).

In particular, given an MMA, the flattening will produce a set of PDDL actions $am_1, .., am_n$, where each am_i $(i : 1..|a.mods|)$ models the behavior of a when the modality $m_i \in a.mods$ is selected. Analogously, for the opposite conversion it is sufficient to create, for each PDDL action a, a corresponding MMA action a' such that the execution modality of a' is set to meet the numeric preconditions and effects of a. Then, we can apply the Definition 4 to group all the MMAs A which has the equivalent abstraction.

This mechanism is employed for solving the numeric planning problem described in Chapter 4.

3.3.1 Action Applicability

Similarly to the numeric planning setting we have that:

Definition 5 (Action Applicability in an Execution Modality)**.** *Given a state S, an MMA a is said to be* applicable *in S with modality $m \in a.mods$ iff:*

- *$a.propPre$ is supported by S_{prop}, meaning that $S_{prop} \vdash propPre$, and*
- *the numeric conditions associated with m, i.e., $a(m).numPre$, are satisfied in S_{num}.*

where numeric and propositional conditions are supported according to the following:

Definition 6 (State and Conditions)**.** *Given a set of propositional conditions and numeric comparisons C (e.g. a propositional precondition and a numeric precondition), and a state S we have that:*

- *S_{prop} satisfies C_{prop} iff (i) $S_{prop} \subseteq$ in C, and (ii) S_{prop} is not undefined;*
- *S_{num} satisfies C_{num} iff (i) S_{num} is consistent (in the mathematical sense) to each comparison present in C_{num}, and (ii) S_{num} is not undefined;*
- *S satisfies C iff both S_{prop} satisfies C_{prop} and S_{num} satisfies C_{num}.*

Of course, given a state S and an MMA action a, it is possible that just a subset of modalities in $a.mods$ are applicable in S.

By reasoning on the multi level nature of the MMA action we can identify different levels of applicability. The applicability condition previously defined indeed requires that the action is applicable with respect to a specific modality. By releasing such further constraint, we can more generally say that:

Definition 7 (Action Applicability)**.** *Given a status S, an action a is applicable iff:*

- *$a.propPre$ is satisfied in S_{prop}, meaning that $S_{prop} \vdash propPre$, and*
- *$\bigvee_{m \in a.mods}(a(m).numPre$, are satisfied in $S_{num})$.*

Meaning that, given an action which is propositionally applicable in a state S, it is also numerically applicable iff at least one of its modalities is applicable in S.

Of course it may be the case that more than one modality is satisfied in a given state. Intuitively, this is an appealing property that, as we will see in Chapter 9 may be used to perform reactive control over the rover parameters configuration. In particular in Chapter 8 we will introduce the notion of *safe execution modalities* through which the system can assure that only particular modalities are actually safe to be executed.

3.3.2 Effects of the Action Application

Given a state of the system S and an MMA a as presented above, the application of a in S with modality m, identified by $S[a(m)]$, produces (deterministically) a new state S' in the following way:

- S' is initialized to be S
- each atom presents in $eff^+(a)$ is added in S' (whether it is not already present)
- each atom presents in $eff^-(a)$ is removed from S' (whether it is present)
- each numeric fluent f involved as a second term of one of the numeric effects of m (i.e. $(< op, f, exp >)$), will be modified according to exp and op, meaning that it can be increased, decreased or it could be set to be a completely new value.

Above, eff^+ and eff^- indicate the *add* and the *delete* list of a.

Similarly to the semantics for actions defined in [42], the application of the numeric effects of a modality into a state of the system depends on the state of the system before the action application. The right value, exp, has a specific evaluation in the state in which the action is applied. Thus, since the effects only depends on the past values of numeric fluents, the order of the application of such fluents does not matter.

Some propositional and numeric fluents, however, could be not involved in the action application; we assume that the values of such numeric fluents persist after the whole action execution (frame axiom).

The state resulting from a non applicable action is undefined by definition. However, considered that a state consists of two separate parts, i.e. the propositional and the numeric one, the result can be either undefined for both subparts or undefined for just one of them. That is, if the action a in a given modality m is propositional applicable in a state S the resulting propositional part of the state S' is defined. Otherwise it will be not. Analogously, for the numeric precondition of m.

The next two sections introduce two small examples of how a multi modality action can be encoded by using a pseudo PDDL code augmented with the use of modalities, that is in the Planetary Rover Domain and in the *ZenoTravel* Domain.

3.4 The Planetary Rover Domain and the role of the MMA

This Section introduces a space exploration scenario, where a mobile robot (i.e., a planetary rover) is in charge of accomplishing exploration tasks.

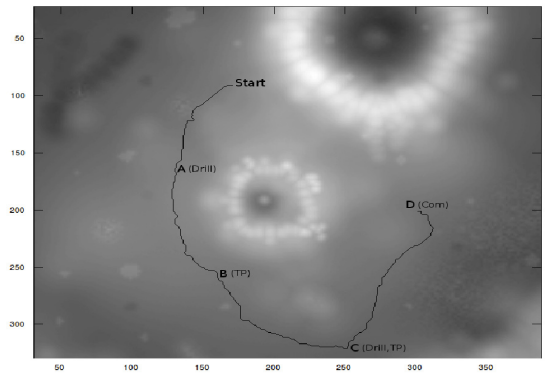
The domain involves the presence of many resources that should be carefully managed. Planetary rovers have limited amount of power and memory therefore the plan being executed has a very large set of constraints to preserve all along the mission.

A typical mission requires to cover a number of scientifically interesting sites: the action domain includes navigation actions as well as exploratory actions that the rover has to complete once a target has been reached; for instance the rover can:
- drill the surface of rocks;
- collect soil samples and complete experiments in search for organic traces;
- take pictures of the environment.

All these actions produce a certain amount of data which are stored in an on-board memory of the rover until a communication window towards Earth becomes available. In that moment the data can be uploaded; see [53] for a possible solution tackling the communication problem in a space scenario.

For example, a possible daily plan could involve: `drive(A,B); drill(B); drive(B,C); tp(C); drive(C,D); drill(D); tp(D); drive(D,E); com(F)`. This plan is graphically represented in Figure 3.3 where a map of a portion of the Martian soil is showed. [5]

[5]In the picture, different altitudes are represented in a grey scale where white corresponds to the highest altitude, and black to the lowest



Capital letters A, B, C, D denote the sites the rover has to visit. Each site is tagged with the actions to be performed when the site has been reached: DRILL refers to the drill action, TP to take picture, and COM to data transmission. More actions can be done at the same site, see for instance target C. The black line connecting two targets is the route, predicted during a path planning phase, the rover should follow during a navigate action.

Figure 3.3: Planetary Rover :: An example of daily mission plan.

For the sake of the discussion, depending on the strategies we will explain throughout the thesis, we can focus just on a subset of the actions defined above. In the rest of the work we will denote this scenario as the *PlanetaryRover* domain.

```
(:action drive
  :parameters (r1 l1 l2)
  :modalities (safe, normal, agile)
  :precondition (and (reachable l1 l2)(in r1 l1))
      (safe: ((>= (power r1) (f_pwr_safe(l1, l2)))
      (normal: ((>= (power r1) (f_pwr_normal(l1,l2)))
      (agile: ((>= (power r1) (f_pwr_agile(l1, l2))) )
  :effect (and
      (in r1 l2)
      (not(in r1 l1)))
      (safe:
         (decrease (power r1)(f_pwr_safe(l1, l2)))
         (increase (time) (f_time_safe(l1, l2))) )
      (normal:
         (decrease (power r1)(f_pwr_normal(l1, l2)))
         (increase (time) (f_time_normal(l1, l2))))
      (agile:
         (decrease (power r1)(f_pwr_agile(l1, l2)))
         (increase (time) (f_time_agile(l1, l2))) )
  ))
```

Figure 3.4: The augmented model of a `drive` action.

Figure 3.4 shows how we express a drive action by using the MMA model. The action `drive (r1, l1, l2)` requires rover `r1` to move from location `l1` to location `l2`[6]. `:modalities` introduces the set of modalities associated with a `drive`; in particular, we express for this action, three alternative modalities:

[6]For clarity reason we report an example of fully instantiated action. However we can adopt action schema as well

- `safe`: the rover moves slowly and far from obstacles;
- `normal`: the rover moves at its cruise speed and can go closer to obstacles;
- `agile`: the rover moves faster than `normal`.

The `:precondition` and `:effect` fields list as usual the applicability conditions and the expected effects, respectively, and are structured as follows: first a propositional formula encodes the precondition/effect of the action, then for each modality listed in `:modalities` a numeric expression is specified; such a numeric expression is used to model the amount of resources and time required (precondition) or consumed (effect) by the action when performed under that specific modality. The numeric part of the model (Mods) of the action is hence distributed in the preconditions and in the effects. For instance, in the preconditions `(reachable l1, l2)` and `(in r1, l1)` are two propositional atoms required as preconditions of the action. These two atoms must be satisfied independently of the modality actually used to perform the `drive` action. While the expression `(safe -> (>= (power r1) (f_pwr_safe(l1,l2))))` means that the modality `safe` can be selected when the rover's power is at least greater than a threshold given by the function `(f_pwr_safe(l1,l2)))`, which estimates the power required to reach `l2` from `l1` in `safe` modality. Analogously, `(safe->(decrease(power r1)(f_pwr_safe(l1, l2)))` describes in the effects how the rover's power is reduced after the execution of the drive action. More precisely, we have modeled the power consumption as a function depending on the duration of the drive action (computed considering distance and speed) and the average power consumption per time unit given a specific modality. For instance, in safe modality, the amount of power consumed is estimated by the function `(f_pwr_safe(l1, l2))`, corresponding to `(safe_cons * (dist(l1,l2)/safe_speed)`, where `safe_cons` and `safe_speed` are the average consumption and the average speed for the safe modality, respectively, while `dist(l1,l2)` is the distance between the two locations `l1` and `l2`.

Finally, note that in the numeric effects of each modality, the model updates also the fluent `time` according to the selected modality. Also in this case, the duration the action is estimated by a function associated with each possible action modality.

Analogously to the drive action, we modeled modalities also for the Take Picture (TP) and the Communication. For TP we have the low (*LR*) and high (*HR*) resolution modalities, whereas for the Communication we assume to have two different channels of communication: *CH1* with low overall `comm_cost` and low bandwidth, and *CH2* with high overall `comm_cost` but high bandwidth.

For the Take Picture action we assume that the rover is equipped with a photo camera which provides two configurations of parameters. The former corresponds to the HR modality and allows a better resolution of the image while consuming a greater amount of memory; we set the HR and the LR modality to consume 2 and 1 unities of memory, respectively.

As anticipated before, also the communication action involves two different modalities; these mainly differ in the way they impact the duration of the action. In particular, the time spent by the action depends on the bandwidth at disposal. We assumed that the communication bandwidth is 1 and 2 units per seconds respectively for *CH1* and *CH2*. The complete model of the domain is reported in Appendix C [7].

3.5 The ZenoTravel Domain in MMA

Another example of multi modality action can be done by considering the *ZenoTravel* domain. The *ZenoTravel* domain is a logistic scenario used in the automated planning competitions for testing the ability of planners in dealing with numeric fluents [8].

The *ZenoTravel* domain involves a set of planes in charge of transporting people among a set of different locations; action models make use of both propositional and numeric fluents.

From our point of view, the *ZenoTravel* domain is quite interesting since in the original version of the model there are two actions (*fly* and *zoom*) that have exactly the same propositional effects, but different numeric effects. Indeed, the two actions have different impacts on the use of resources (in particular on the numeric fluents *fuel* and *total_fuel_used*). These two actions have different preconditions, but it is worth noting that the propositional preconditions are exactly the same (the airplane must be in the source airport) and the only differences in preconditions are the ones related to numeric fluents. It is hence quite interesting to observe that these two actions can be perfectly captured in our approach with a single action fly with two modalities *cruise* and *zoom*. In particular, the multi-modality action schema of fly, in a pddl-like language, is shown in Figure 3.5.

It is easy to see that the propositional preconditions and effects are shared among all the modalities, while the preconditions and effects concerning numeric fluents are specific for each modality.

To make the domain more challenging, in the experimental phase using this scenario, we added to the original model further numeric fluents. We introduced the *time_spent* fluent, whose increase, after the execution of action, depends not only on the action but also on the modality (see Figure 3.5). We also added two modalities for the *board* and *debark* actions, namely, *normal* and *express* which take different amounts of time (shorter for *express*) and different costs (lower for *normal*).

[7] To keep the explanation simple we omit some further fluents that have been used in the actual domain. Particularly this further information have been added to make more interesting the benchmark suite for our experimental analysis

[8] http://planning.cis.strath.ac.uk/competition/

```
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :modalities {cruise, zoom}
 :precondition (located ?a ?c1)
 :numPrecondition
         (cruise:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (cruise-burn ?a)))
         (zoom:
            (>= (fuel ?a) (* (distance ?c1 ?c2) (zoom-burn ?a))))
 :effect (and (not (located ?a ?c1))
              (located ?a ?c2)))
 :numEffect
         (cruise:
            (and(increase (total-fuel-used) (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (decrease (fuel ?a) (* (distance ?c1 ?c2) (cruise-burn ?a)))
            (increase (time-spent) (/ (distance ?c1 ?c2) (cruise-speed ?a)))))
         (zoom:
            (and(increase (total-fuel-used) (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (decrease (fuel ?a) (* (distance ?c1 ?c2) (zoom-burn ?a)))
            (increase (time-spent) (/ (distance ?c1 ?c2) (zoom-speed ?a)))))
```

Figure 3.5: Multi Modality Action template for the `fly` action in the *ZenoTravel* domain.

3.6 Conclusion

In this Section we have introduced the notion of a MMA. The main idea is that in many domains a same action can be actually performed in several *execution modalities*. For this reason the MMA provides two different levels of abstraction. The higher captures the main qualitative preconditions and effects of the action, while the lower specifies, for each modality, which are the requirements and the effects over a given set of resources.

The Section formalized the notion and reported some concrete example of MMA in a PDDL-like language.

The MMA notion is fundamental for the comprehension of the next Chapter, which introduces the *multi modality plans*.

Chapter 4

Multi Modality Plans: the Dynamic Modality Allocation Problem

This Chapter introduces the notion of a Multi Modality Plan (MMP) which is a total ordered set of instantiated MMAs. The Chapter discusses how computing and handling an MMP by means of the Multi Modality Planning Problem (MMPP) and the Dynamic Modality Allocation Problem (DMAP). Then the Chapter reports some theoretical properties of the given formulations. At the end, a practical way to compare two plans in terms of stability will be introduced.

4.1 Introduction

In the previous Chapter we have seen how the agent encodes the world state, under which conditions an MMA can be applied and how such an action transforms the state of the system once it has been applied. This Chapter introduces the notion of the *multi-modality plan* (MMP) as a result of a *multi-modality planning task*. Intuitively, the multi modality plan is a sequence of MMAs and is aimed at representing the task the agent is in charge to perform.

The Chapter analyzes two levels of consistency that can be detected by monitoring the MMP during the execution. An MMP can be *valid*, meaning that the execution can go on. Otherwise it can be either *partially valid* or *invalid*, meaning that some kind of recovery mechanism is necessary. In particular, when the plan is just *partially valid* the repair can be performed by solving the Dynamic Modality Allocation Problem. The DMAP is key for the supervision task of the FLEX-RR mechanism reported in Chapter 6.

4.2 Multi Modality Planning Problem

This Section introduces the planning framework of reference for the MMAs introduced in the previous Chapter. In particular, we introduce a Multi-Modality Planning Task, i.e.:

Definition 8 (Multi Modality Planning Problem)**.** *Given the domain* $< U, F, X >$*, a Multi-Modality Planning Problem (MMPP)* Π *is a tuple* $< A, I, G_{prop}, G_{num} >$ *where:*

- A *is the set of possible MMA instances over* $< U, F, X >$*;* [1]
- $I \in 2^F \times \Re^{|X|}$ *is the initial state of the problem;*
- G_{prop} *is the propositional goal; i.e., a partial state over the propositional fluents in* F*;*
- G_{num} *is the numeric goal; i.e., a conjunction of numeric constraints over* X*.*

Note that our notion of MMPP is similar to the Numeric Planning Problem (NPP) proposed by Gerevini et al. [45]; the main difference is that we make explicit the adoption of actions with modalities rather than traditional PDDL actions. It is easy to see, however, that an MMPP can be translated into an NPP; it is sufficient to *flatten* each MMA a in A by generating a set of corresponding PDDL actions as reported in Chapter 3. Analogously, we can perform the opposite conversion: given a plan consisting of PDDL actions, we can build an MMA plan[2].

Let be $S[\pi]$ the state produced as a result of the application of the actions of the plan π starting from the state S, the multi modality plan is formally defined as follows:

Definition 9 (MMPP Solution)**.** *Given a MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, the plan* $\pi = a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})$ *is a solution for* Π *iff:*

i. $I[\pi] \vdash G_{prop}$ *and*

ii. $I[\pi]$ *satisfies* G_{num}

In other words, a sequence π of MMA instances (specified with their modalities) represents a solution for Π when the execution of π transforms the initial state I into a final state $I[\pi]$ such that: (1) the propositional atoms in G_{prop} holds in $I[\pi]$, and (2) the numeric constraints in G_{num} are satisfied.

[1] Since we are interested in plan execution, we consider in this formulation instantiated actions only. The extension for action schema is straightforward

[2] In Chapter 6 we will describe how practically one can invoke a PDDL planner starting from MMAs

In the following definitions, we use the notation $\pi_{i\to j}$ with $i \leq j$ to denote the plan segment starting at the i-th action a_i of the plan and ending at the j-th action a_j; when the right bound is omitted (e.g. π_i) the length of the plan is assumed, that is π_i is equal to $\pi_{i\to|\pi|}$. Moreover, since a plan *segment* is itself a plan, the notation $S[\pi_{i\to j}]$ represents the (intermediate) state S' reached after the execution of the actions $a_i, .., a_j$ from the state S.

The operation for the application of a plan $\pi = a_0(m_0)$,$a_1(m_1)$,..,$a_{n-1}(m_{n-1})$ in a state S is well defined only when each action of the plan is applicable in the state produced by the action before, i.e. each action $a_i(m_i)$ must be applicable in the state $S[\pi_{0\to i}]$. Otherwise the state resulting from the operation is undefined (either in the propositional or numeric sense, see Definition 6).

4.3 On-line execution of Multi Modality Plans

In this Section we will formalize the notion of an executable plan. Intuitively, the notion is introduced to formalize the conditions under which a plan is still consistent with respect to the upcoming situation. Moreover, we will discern two levels of consistency for a given plan. Such a distinction is important in order to understand when and how a Dynamic Modality Assignment Problem arises.

4.3.1 Executable Plans

The actual execution of a plan π in the environment can be threatened by the occurrence of unexpected events such as variations in the resource consumptions, or assumptions that, made at planning time, become invalid at execution time. It is therefore essential to detect, while the plan is still in progress, any unexpected deviation from the nominal expected behavior.

More formally we can say that:

Definition 10. *Given an MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, and a plan* $\pi =$ $\{a_0(m_0)$,$a_1(m_1)$,..,$a_{n-1}(m_{n-1})\}$ *solving* Π*, let* S^i *be the* observed *system state obtained after the execution of the first i actions in* π*, and let* π_i *be the plan segment* $\{a_i(m_i), ..a_{n-1}(m_{n-1})\}$ *still to be performed; we say that:*

- π *is* valid *at execution step i iff* $S^i[\pi_i]$ *satisfies both* G_{prop} *and* G_{num}*; namely, the final state predicted by applying the plan segment* π_i *to* S^i *satisfies both the propositional and numeric terms of the plan goal;*

- π *is* partially valid *at step i iff* $S^i[\pi_i]$ *satisfies* G_{prop} *but not* G_{num}*;*

- π *is* invalid*, iff* $S^i[\pi_i]$ *does not support* G_{prop}*.*

Thus, after the execution of each action in the plan, it is necessary to determine the status (i.e., either *valid*, *partially valid*, or *invalid*) of the plan. For this purpose, we do not consider just the effects of the last performed action, but we also verify whether the propositional goals G_{prop} and the numeric ones G_{num} can be satisfied in the final state reached by π. We do this starting from the observed state S^i of the world in order to capture both the changes caused by the agent, and the changes caused by the possible occurrence of exogenous events.

The assessment of the actual environment state after the execution of an action is, in general, a complex problem since the environment is, in most cases, not fully accessible, and only partial observations about the world are available. The problem of plan execution in partially observable environments has been dealt in [66] [86]. In this paper, however, we assume that the observability level is sufficient for measuring the effects of an action $a(m)$ (i.e., applied with modality m), and for assessing the information present in the precondition set of the next action to execute.

For the sake of explanation let us show a small example of plan consistency in the *Planetary Rover* domain.

An example of partially valid plan

Let us consider a simple problem for the *Planetary Rover* domain whose initial state[3] and the goals statement are reported by Figure 4.1 (left part); whereas the MMA plan provided as solution is reported in the right part of Figure 4.1.

In particular, the goal conditions constrain, on one hand, the use of resources, and on the other hand, specifies that the rover *r1* has to acquire information in sites B and C.

Figure 4.2 shows the final state (i.e. $I[\pi]$), predicting the evolution of the environment according to the model of the actions of the *Planetary Rover* domain defined in Appendix C. Precisely, the rover will complete the plan in less than 50 unities of time, consuming less than 666 unities of power.

At any step of the plan execution, the agent can refine the predictions made by integrating the information coming from the real observations. In a nominal condition, that is, when no exogenous event (or a anomalous action behavior) occurs, it is quite obvious that the final status of the system will be consistent with the prediction made at planning time; thus the current plan of actions still represents a valid solution for the mission at hand, and in our example the rover will arrive to the end of the mission achieving the pre-fixed objectives.

Let us consider a non nominal condition instead, supposing that the *drive* action of step 1 actually consumed more power than expected. It is quite evident

[3]The initial status is the same defined in 3.1. Here we report the definition for convenience

```
/*INITIAL STATE*/

/*propositional info*/
(at (r1,A))
(road (A,B))
(road (B,C))
(road (C,D))
(road (A,D))
/*numeric info*/
(= (distance(A,B)) 15)
(= (distance(B,C)) 35)
(= (distance(C,D)) 45)
(= (distance(A,D)) 10)
(= (power r1) 1000)
(= (memory r1) 4)
(= (time) 0)

/*GOAL CONDITIONS*/

/*proposition goal*/
(info_acquired(r1,B))
(info_acquired(r1,C))
/*numeric goal*/
(> (power r1) 300)
(< (time) 50)
(= (memory r1) 4)
```

```
/*MMA PLAN*/

1: Drive(A,B)(cruise)
2: TP(B)(HR)
3: Drive(B,C)(cruise)
4: TP(C)(HR)
5: Comm(C)(CH1)
```

Figure 4.1: Planetary Rover Domain :: Problem and MMP

```
PREDICTION

/*propositional information*/
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric information*/
(= (power r1) 334)
(= (time) 47)
(= (memory r1) 4)
```

Figure 4.2: Planetary Rover Domain :: Step 0 Prediction

```
PREDICTION

/*proposition goal*/
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric goal*/
(= (power r1) 290)
(= (time) 47)
```

Figure 4.3: Planetary Rover Domain :: Step 2 Prediction

that as a consequence, the prediction made at the beginning of the execution is not reliable anymore and hence has to be refined.

To this purpose, the agent can re-simulate the whole plan execution from the new observed state. Thus, let us imagine that, at step 2 (before the execution of *TP(B)(HR)*), the final status is predicted according to Figure 4.3.

According to definition 10, the plan turns out to be just partially valid as the propositional goals continue to be verified while the numeric goals do not. Indeed the power of the robot is predicted to be less than expected, and this situation is not consistent with the numeric goal conditions.

In a different scenario, the impact of the current observations could be more sweeping. Imagine that, at the same time step and differently to what the agent has assumed at planning time, we notice that the road connecting the site B with the site C is no more available (e.g. because of some obstacle prevents the passage). Then, albeit the next action is still executable, the plan is assessed to be invalid. Indeed the current state, namely S is such that $S[\pi]$ does not satisfies G_{prop}. In particular the propositional preconditions of $DRIVE$ of step 3 would not be satisfied, because of the action instance requires the presence of the road connecting B and C, and hence $S[\pi]$ turns out to be undefined (in the propositional sense), meaning that it does not satisfy any propositional conditions.

4.4 The Dynamic Modality Allocation Problem

In principle, whenever the plan π is no longer *valid*, a replanning mechanism should be invoked in order to find an alternative way to reach the desired goals. As we have seen, however, the clear distinction between the propositional and numeric aspects within an MMA allows us to distinguish between a completely invalid plan (which is no longer executable because some of the required propositional fluents are missing), and a plan which is just propositionally consistent but not numerically, i.e. *partially valid*. The idea is that, while an invalid plan

can only be repaired by means of a (possibly expensive) replanning step, it may be the case that a partially valid plan can be repaired more effectively by reconfiguring its actions. That is, we try to reuse the effort made for the synthesis of π by adjusting the way in which the actions in π will be performed. We therefore propose to repair a *partially valid* plan not via a replanning step but via a reconfiguration phase, and to do this, we introduces the notion of Dynamic Modality Allocation Problem (DMAP), and we study some interesting properties.

According to Definition 10 and by reasoning on the MMPP, the Dynamic Modality Allocation Problem can be formally defined as:

Definition 11 (The Dynamic Modality Allocation Problem (DMAP))**.** *Let* $\pi = a_0(m_0), .., a_i(m_i), .., a_{n-1}(m_{n-1})$ *be a solution for the MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, and let* S^i *be the last observed state of the environment (i.e., we are at the i-th execution step).*

The Dynamic Allocation Problem Φ *is a tuple* $< \pi, i, S^i, G_{num} >$ *such that:*

- π *is an MMP whose first i actions have been already performed,*
- π *is partially valid at step i,*
- S^i *is the observed state of the environment after the execution of action* a_{i-1}*,*
- G_{num} *is the set of numeric constraints to be fulfilled.*

Definition 12 (Solution of a DMAP)**.** *The solution (if any) of a DMAP is a new plan*
$\pi' = a_0(m'_0), .., a_i(m'_i), .., a_{n-1}(m'_{n-1})$ *such that:*

- *for each action* a_j*,* $0 \leq j < i$*,* $m'_j = m_j$*: the reconfiguration cannot change the modality of an action that has already been performed;*
- $S^i[\pi'_i] \vdash G_{num}$*: the new assignment of modalities satisfies the numeric portion of the goal.*

It is worth noting that the solution π' of a DMAP has the same sequence of actions as π. The two plans, π and π', just differ each other in the modalities associated with the actions not yet performed. Note also that, since the input plan is *partially valid*, the solution of DMAP will differ from the original setting of modalities at least for one of them; that is, for each action a_j, $i \leq j < n$, $m'_j \in a_j.mods$ can differ from m_j;

Moreover it is easy to see that if M is the average number of modalities for the involved actions in π, the search space for a DMAP at step i is $M^{|\pi_i|}$. As long as the execution proceeds, the number of actions to consider decreases; thus the search space decreases too.

The problem is 'dynamic', since it may arise at any step of the plan execution, and assignments made at a given step could be reconsidered afterwards.

Having solved the DMAP problem we are sure that the new MMA plan is a valid solution for the overall MMPP; that is:

Theorem 1. *Let π' be a solution for the DMAP $\Phi =< \pi, i, S^i, G_{num} >$, where π is* partially valid *at step i wrt the MMPP Π, then π' is valid at step i.*

Proof. To show that π' is valid we have to prove that $S^i[\pi'_i]$ satisfies both G_{num} and G_{prop}. The former comes directly from the definition of a solution for the DMAP, whereas the latter follows by the fact that π' inherits the causal structure of π (i.e., the qualitative/propositional part of the plan does not change when the solution is computed). For this reason π' is computed from a *partially valid* solution and hence $S^i[\pi'_i]$ satisfies G_{prop}. □

It is easy to see that the solutions space of a DMAP is a subset of the solutions space defined by MMPP. Therefore a solution for the DMAP, if any, is also a solution for the MMPP. However, the opposite does not hold, namely when the DMAP has no solution, the MMPP might instead have a solution. For the sake of clarity Figure 4.4 clarifies the situation.

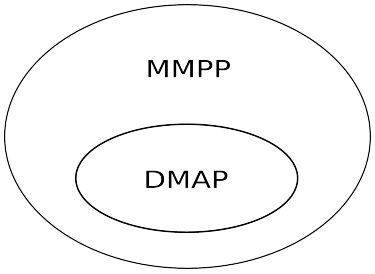


Figure 4.4: The relation between the space of the solutions of $MMPP$ and $DMAP$

The relation between an MMPP and a DMAP is still more evident by studying the worst case complexity. It is worth remembering for this purpose that the automated planning is hard from a computational point of view. Indeed the only classical fragment (i.e. STRIPS) has been showed to be PSPACE-complete (see [18])) and, to the best of our knowledge, despite there are no complexity results on the numeric extension of the classical paradigm, the numeric planning and hence the MMPP is supposed to be at least complex as a STRIPS problem.

On the other hand, the characterization given in this Chapter, the DMAP, is less complex. In fact:

Proposition 2. *The Dynamic Modality Allocation Problem is an NP problem*

The proof of the NP membership trivially comes from the polynomial conversion mechanism reported in Chapter 5. There we will show that a DMAP

can be transformed in a standard CSP. As well known, a CSP is a classical NP-complete problem; so the resolution of the DMAP is not much harder of an NP problem, i.e. the DMAP is in NP.

Given the proposition above we can also state that:

Observation 3. *If PSPACE $\supset$ NP and MMPP $\supseteq$ STRIPS then DMAP $\subset$ MMPP, meaning that solving an instance of the DMAP is simpler than solving an MMPP.*

It is easy to see here that, by hypothesis, we have that MMPP $\supseteq$ PSPACE $\supset$ NP $\supseteq$ DMAP hence the thesis, MMPP $\supset$ DMAP, is a direct consequence. The first hypothesis of the observation (PSPACE $\supset$ NP) is supported by the fact that (i) we know that $NP \subseteq PSPACE$ ([7]) and (ii) it is widely assumed that the converse does not hold, i.e. $PSPACE \nsubseteq NP$.

These theoretical observations gives us a formal guarantee that, in the worst case, reconfiguring is simpler than replanning from scratch and the result will be still more evident in the experimental session reported in Chapter 7.

4.5 Comparing solutions found by DMAP and MMPP

In many domains, the impact of the decisions taken by repair mechanisms for restoring the plan execution represents a critical issue. Since new decisions might diverge from the ones taken at planning time (e.g., different actions or also just a change in the order of the actions execution), the behavior of the agent might become difficult to predict for an external observer, as for instance a human supervisor. In addition, when the agent has to cooperate with others, repairing the plan with a minimum amount of changes is a desirable property to make joint activities feasible, as pointed out by other authors too ([60],[44], - see Chapter 2.2.3 for a detailed discussion). The problem is that agents could have provided commitments and agreements on the initial decisions. For this reason, the revision of the "local" choices could have a significant impact from a global point of view (see for instance [102]).

By observing the plans space of the DMAP and the MMPP it is possible to evaluate the difference of the kind of solutions that can be found by the two approaches. The following theorem better characterizes such a difference. In a nutshell, when the DMAP fails, the solutions to the new MMPP must differ from the original plan for at least one action.

Theorem 4. *Let π_i be a partially valid plan at step i, and let us assume that the DMAP $\Phi =< \pi, i, S^i, G_{num} >$ has no solution. Let π' be a solution for the new planning problem $\Pi' =< A, S^i, G_{prop}, G_{num} >$. Then π' satisfies at least one of the following statements:*

- *there is at least an MMA in π' which does not belong to π_i.*
- *there is at least an MMA in π_i which does not belong to π'.*
- *the order of MMAs in π' differs from π_i, i.e. there are at least two actions a and b where both $a,b \in \pi'$ and $a,b \in \pi_i$ such that if $a \prec b$ in π' then $b \prec a$ in π_i.*

Proof. The proof proceeds by contradiction. Let us assume that the solution found for Π' is such that none of the three statements hold. If this is the case the solution π' will have exactly the same structure of π_i in terms of MMAs contained as it has no different action (first and second statements) and the order is the same (third statement). It is easy to see hence that the only way in which π' differs from π_i is in the modality assigned to each action.

Moreover, by definition of Π' we know that if π' is a solution for Π', then $S^i[\pi']$ satisfies both G_{prop} and G_{num}. Of course, if π' has the same structure of π_i it means that there will be an assignment of modality in π_i such that $S^i[\pi_i]$ satisfies G_{num}. On the other hand, by hypothesis, we know that the DMAP did not found any assignment of actions modality for the piece of plan π_i which obviously is in contradiction with the fact that there is an assignment in π_i such that $S^i[\pi_i]$ satisfies G_{num}. □

Theorem 4 shows that the DMAP actually complements the replanning from scratch: when there is no solution for a DMAP, the original plan structure cannot be preserved, and some actions have to be added, removed, or put in a different order in the repaired plan. This satisfies our intuition that, in order to keep the plan as stable as possible, it is helpful to try resolving a DMAP first, and only when this has no solution then attempt to re-plan from scratch.

Relying on Theorem 4, our hypothesis is that the reconfiguration produces, on average, more stable repaired plans than a replanning from scratch.

However there may be cases for which the reconfiguration needs many changes in the set of re-configurable actions, while the simple adding of an action (achievable via MMPP) may handle the unexpected contingency. For this reason, to confirm, or negate such a hypothesis, we propose here a way to evaluate the *stability* of a repaired plan, based on the notion of *distance* between two plans (i.e., the original and the repaired plan). Doing so, we will have a metric to evaluate the approach in the experimental sessions (Chapter 7).

4.5.1 Stability

Our stability measure relies on the *distance* $D(\pi', \pi)$ between the new (repaired) plan π' and the original plan π. Such a measure is inspired to the well-known Levenshtein's string distance [58].

We compute such a distance as the minimum cost for transforming π' into π, where a transformation consists of a sequence of operations, each of which has a positive cost. The operators we consider are:

- inserting (*add*) and removing (*del*) an action in π', with cost α
- replacing (*remod*) the modality of an action in π', with cost γ
- swap (*swp*) the order of two consecutive actions in π', with cost θ

It is reasonable to assume that $\gamma \ll \alpha < \theta < 2 * \alpha$; that is, changing a modality is less expensive than inserting/removing an action, that in turns is cheaper than swapping two actions. Of course, swapping two consecutive actions is less expensive than adding and removing an action.

Thus, the transformation $\tau[\pi', \pi]$, transforming π' into π, involves a certain number $adds_\tau$ of insert operations, $dels_\tau$ delete operations, $remods_\tau$ replace modality operations, and $swps_\tau$ swap operations.

The cost of τ is therefore computed as:

$$cost(\tau[\pi', \pi]) = \alpha * adds_\tau + \alpha * dels_\tau + \gamma * remods_\tau + \theta * swps_\tau$$

Let T be set of all the possible transformations of π' into π, the *distance* $D[\pi', \pi]$ between the two plans is the cost of the *cheapest* transformation in T:

$$D[\pi', \pi] = \min_{\tau \in T} cost(\tau[\pi', \pi])$$

The evaluation of the distance $D[\pi', \pi]$ can be done via a dynamic programming procedure, similar to one proposed by Levenshtein, that allows us to find the minimal distance without computing the set T first.

Having this notion of distance between plans, we can define the stability of π' w.r.t. π. Intuitively, the stability has to be maximum when no change happens, i.e. π' equals π, and minimum when the two plans are completely different.

$$stability(\pi', \pi) = \frac{cost(\tau_{trv}[\pi', \pi]) - D[\pi', \pi]}{cost(\tau_{trv}[\pi', \pi])}$$

where $cost(\tau_{trv}[\pi', \pi])$ is used as a reference cost; it corresponds to the more expensive transformation, τ_{trv}[4], which first removes all the actions in π', and then inserts all the actions in π to π'. The formula has its maximum/minimum in 1/0. Thus, when $D[\pi', \pi]$ is close to the cost of τ_{trv}, this suggests that the new plan π' is substantially different from the original one, and hence we compute a low grade of stability (close to zero). On the other hand, when $D[\pi', \pi]$ is significantly lower than the cost of τ_{trv}, the two plans π and π' are very similar, and we compute a high grade of stability (close to one).

[4] τ_{trv} is the most expensive transformation we can find with our dynamic programming procedure

Computing the plan stability: A simple example

Let us consider a plan from the *ZenoTravel* Domain (see Appendix C.2) and let us measure the stability in case the plan is reconfigured (via DMAP), Figure 4.5, or it is re-generated from scratch (via MMPP), Figure 4.6.

```
ORIGINAL:                    REPAIRED:
debark_P2_F1(normal)         debark_P2_F1(normal)
board_P3_F1(normal)          board_P3_F1(normal)
fly_F1_A2_A3(zoom)           fly_F1_A2_A3(cruise)
debark_P1_F1(normal)         debark_P1_F1(express)
debark_P3_F1(normal)         debark_P3_F1(express)
```

Figure 4.5: Reconfiguration via DMAP

```
ORIGINAL:                    REPAIRED:
debark_P2_F1(normal)         debark_P2_F1(express)
board_P3_F1(normal)          board_P3_F1(express)
fly_F1_A2_A3(zoom)           refuel_F1_A2(default)
debark_P1_F1(normal)         fly_F1_A2_A3(zoom)
debark_P3_F1(normal)         debark_P1_F1(express)
                             debark_P3_F1(express)
```

Figure 4.6: Replanning via MMPP

Given that α (the insertion/deletion cost) equals to 5, γ (modality replacement cost) to 1 while θ (the swap cost) is set to 6, we have that:

- DMAP: the trivial transformation costs 50 (the cost for 5 deletions and 5 insertions), the distance between the original and the repaired plan is 3, and hence the stability grade is 0.94, meaning that the two plans are very close to each other.
- MMPP: the trivial transformation costs 55, the distance between the original and the repaired plan is 9 and hence the stability grade is 0.83.

This means that the plan repaired via replanning is slightly less stable than the plan repaired by solving the DMAP.

In literature, as we have seen in Chapter 2, the stability metric has been firstly introduced in [60]. In this work, however, they do not capture the concept of an execution modalities. Instead the stability is computed has a measure accounting only the presence, or the absence, of actions among the two plans considered. For this reason their metric does not meet our purpose in that we have no means to differentiate the impact caused by a change of modality and the one by a change of an action.

Another metric for evaluating the distance between two plans has been introduced recently in [77]. However, also in this work there is no distinction among action and modalities; rather they establish the distance among the two

plans by considering their causal structure. Intuitively, the more different the causal structure of the two plans is, the larger the distance between these two plans will be. The intuition could be useful for our purpose as well. However, it should be studied how to combine the causal structure with the concept of execution modality, above all considering the interdependence arising with the numeric/resource counterpart.

4.6 Conclusions

In this Chapter we have presented the formal problems of reference for dealing with and managing the concept of MMA. We have seen that an MMP, i.e. a plan of MMAs, is the result of the general MMPP. Therefore an MMP can be generated by using a general numeric planner system properly adapted to our purposes.

More important, the Chapter individuates three different levels of validity for a plan that can be detected online, during the plan execution. In particular, when the plan is just *partially valid*, the revision can be performed via the DMAP. The DMAP is a reformulation of the general MMPP; the reformulation allows to state the problem of repair as a reconfiguration problem.

The DMAP characterization has several theoretical benefits which let us to believe that the reconfiguration can be more efficient than a replanning from scratch and, moreover, the plan structure can be preserved despite the adaptation. As we will see in Chapter 7, these properties will be positively verified experimentally, too.

Finally the chapter introduces a new stability metric for the purpose of evaluating the impact of the repair mechanism on the plan, in the context of MMAs.

Chapter 5

Modeling the DMAP within a CSP

This Chapter introduces the CSP conversion exploited in order to represent and solve the Dynamic Modality Allocation Problem (DMAP). The given CSP formulation will be used also as a solid framework for handling the MMA plan execution process in general. As we will see in Chapter 6, the CSP engine plays an essential role in the FLEX-RR supervision task.

5.1 Introduction

So far, we have pointed out that a plan can be made *partially valid* or *invalid* by unexpected contingencies occurring during the execution phase. We have also suggested that, when the plan becomes just *partially valid*, the agent can try to solve a DMAP to reconfigure action modalities. On the other hand, when the plan becomes *invalid*, the space of modalities is not sufficient to accommodate the contingency; therefore the agent has to find a new course of actions (i.e. replanning from scratch). In particular, we have presented in a declarative way the Dynamic Modality Allocation Problem (DMAP).

In order to handle the DMAP from a computational point of view, we translate the given formulation in a Constraint Satisfaction Problem. Indeed, the constraint satisfaction problems (CSP) and the solvers developed so far have reached an high degree of maturity ([9]) and have been successfully applied in a very large range of applications including planning and scheduling ([36] [59]). For a thorough CSP description see [32].

The adopted CSP formulation is mandatory for the resolution of the DMAP. In addition, since a DMAP could arise many times during the plan execution, we have further developed a mechanism for translating the plan model into a CSP

model just once, and reusing the same CSP model for the different occurrences of DMAP. In this way we do not need to re-build the CSP model completely from scratch having a solid overview on the plan execution process in general. For details on this aspect see Chapter 6.

In the next two Sections we will describe the fundamental steps to build the CSP model representing a specific instance of DMAP; more precisely the discussion is organized to show firstly what are the variables involved within the CSP and after which we will explain how such variables have to be constrained. The constraints binding the variables of our model are mandatory to capture only solutions that are consistent with the DMAP formulation.

The translation we will propose aims at generating a general CSP encoding for the DMAP, so we can in principle use an CSP engine[1]. As we will see in Chapter 7, even without the adoption of ad-hoc search strategies, we can obtain very good performance. However, a careful translation is desirable. For this reason, we have implemented some enhancement to speed up the resolution process (see Section 5.4).

5.2 CSP Transformation: Encoding the DMAP

The schema guiding our CSP construction is similar to the translation proposed for the classical planning setting in [36, 59].

In a classical planning setting the CSP model requires two sets of variables: one to model the actions selection, and the other one to model the states generated by the action execution. Analogously, the encoding of a DMAP into a CSP representation relies on two sets of variables. In particular:

i. $MODs$ is the set of modality variables: the solution of a DMAP is in fact an assignment of modalities to the plan actions, thus the model has to focus its attention on modality variables, instead of action variables;

i.i. $NUMs$ is a set of state variables just encoding the numeric fluents of the original DMAP problem, while propositional fluents are not considered.

The former set of variables captures the modalities to choose. As reported in Definition 3, each action encompasses a set of modalities of execution, and only one of these shall be actually selected for the execution. By modeling each modality with a single variable, we automatically support the previous requirement. Indeed, a specific CSP solution will assign just a value for each modeled variable.

As far as it is concerned by $NUMs$, similarly to CSP-based planning (based on a graphplan-like structure [1]), we duplicate the state variables as many times

[1] In our system we adopted the Choco Solver. For details see `http://www.emn.fr/z-info/choco-solver/`

as there are "levels" in the plan π[2].

One of the advantages of the DMAP is that it does not require to model the propositional aspects of the planning problem. For this reason the CSP set of state variables, i.e. $NUMs$, just maps the numeric fluents of the state of our system, i.e. S_{num}.

Of course, the CSP representation also involves a number of constraints. In particular in our context the constraints bind modality variables and state variables among each other; each constraint depends on the model of the action and the specific instance of the problem.

In other words, the CSP model binds the state variables at level l with:

- the (numeric) preconditions of the modality variable associated with action a_l;
- the (numeric) effects of the modality variable associated with action a_{l-1}.

State variables at level 0 are set to represent the initial state of the world, while variables at the last level are also constrained with the numeric part of the plan goal.

In our formulation the only decisional variables are the modality variables as the state variables are computed as side effects of modality selections.

For the sake of clarity, Figure 5.1 summarizes the variables and the constraints involved in a generic plan of length n; in particular the figure describes:

- State Variables $NUMs$ in the *snapshot-layers*. For each numeric fluent N_i in X, a variable V_i^j (with j from 0 to $|\pi|$) is added in $NUMs$.
- Modality Variables $MODs$ in the *action layers*. For each MMA a_k belonging to the plan, a modality variable mod_k, taking values in $mods(a_k)$, is included in $MODs$.

Intuitively, the superscript j of a variable V_i^j in $NUMs$ represents the execution step the variable refers to. Thus, all the numeric variables labeled as 0 represent the initial state; whereas variables labeled as $0 < j \leq |\pi|$ represent (the numeric portion of) the state after the execution of action a_{j-1} with modality mod_{j-1}.

[2]Of course, since we are not solving a planning problem, but a DMAP, we do not need the step of graph expansion as the sequence of actions is fixed and already known.

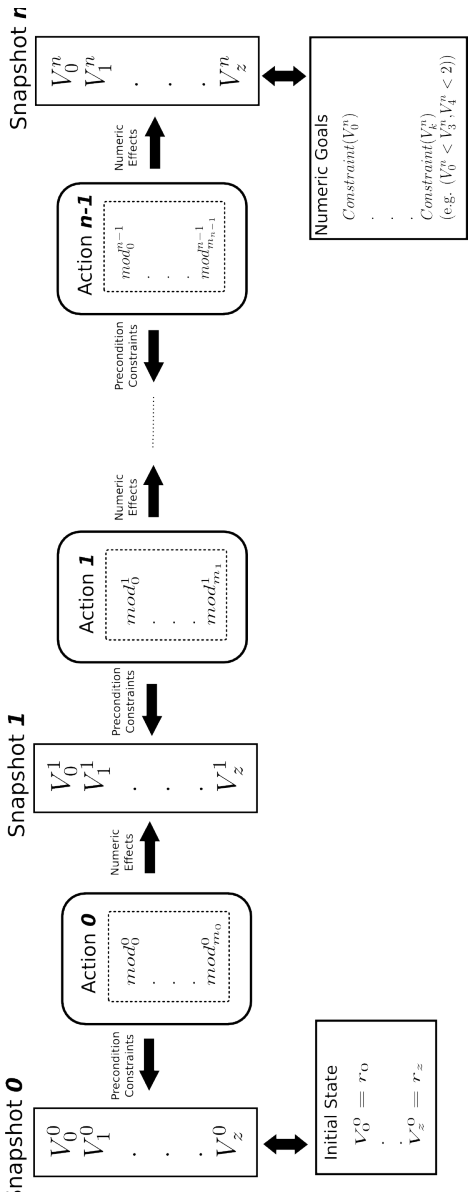


Figure 5.1: A CSP representation for a generic MMA plan of length n in a problem with z numeric fluents and k goal constraints.

5.3 CSP Constraints Formulation

Section 5.2 introduced the variables of the CSP formulation for the DMAP problem, and some clues concerning the constraints that have to be built around the variables of our problem. However, the formulation of the preconditions and of the effects in the action model can be complex and may involve many numeric fluents interacting among each other. For this reason, this Section describes how the constraints binding the variables in $MODs$ and $NUMs$ are built. Let us remember the reader that this step is mandatory for guaranteeing to obtain only assignments of modalities that are consistent with the model of actions and the problem at hand.

Init and Goal Status Constraints

First of all, let us see how the initial state and the goal conditions of a DMAP can be represented in terms of variables and constraints within a CSP. Indeed, the first and the last snapshot of our CSP representation must be constrained in order to be consistent with the state in which the i-th action applies and the goal conditions, respectively, of the DMAP formulation.

The initial setup of the CSP formulation translates a specific instance of the DMAP, i.e. $\Pi =< \pi, 0, I_n, G_{num} >$.

1. *Initial State Constraints.* For each *numFluents* in I_n of the form $N_i = val$, the corresponding CSP variable $V_i^0 \in NUMs$ is constrained to assume the value val [3]:

$$\forall\ N_i = val\ \in\ I_n,\ \ CSP.addConstraint(V_i^0 = val)$$

   where val is a constant in $\Re$.

2. *Goal Constraints.* For each comparison of G_{num}, which has the form $C :< exp, comp, exp' >$, where exp and exp' are linear expressions mentioning numeric fluents in X, and $comp$ belongs to $\{<, \leq, =, \geq, >\}$, we create a corresponding comparison mentioning the variables in $NUMs$ at step n, where $n = |\pi|$:

forall the C $\in G_{num}$ **do**
  CSPExp $\leftarrow Subst(C.exp, X, V^n)$
  CSPExp' $\leftarrow Subst(C.exp', X, V^n)$
  $CSP.addConstr(CSPExp, C.comp, CSPExp')$

[3] Note that, an in initial state is a complete state, meaning that there is no numeric fluent with an undefined value

In the transformation reported above, the procedure *Subst* substitutes the variables in the goals definition with the variables of the last snapshot of the CSP Model. The result is a CSP expression stating how the variables of the CSP Model relates to each other[4].

Precondition Constraints

As anticipated in the previous Section, an MMA a is said to be applicable in a given state S with modality m only when the numeric preconditions associated with m are satisfied (assuming that the propositional preconditions are satisfied too) in S. To encode the applicability condition within the CSP model, we have to bind each MMA a_i with the CSP variables in V^i; that is, state variables encoding the state in which action a_i has to be executed. Therefore, for each MMA a_i in the plan, we add these constraints as follows:

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the m *in* $a_i.Mods$ **do**
    forall the C $\in numPre(m)$ **do**
      CSPExp := $Subst(C.exp, X, V^i)$
      CSPExp' := $Subst(C.exp', X, V^i)$
      CSP.addConstr($(mod_i = m) \rightarrow$
      $(CSPExp, C.comp, CSPExp')$)

Namely, for each modality m associated with a_i, we consider every comparison C in $numPre(m)$, and translate it into a new expression in terms of CSP variables in V^i, similarly to what we have shown for the translation of the goal constraints. The new expression is therefore added to the CSP model.

Effects and Frame Constraints

The execution of an action changes the system state according to its model. In general, however, just a portion of the system state is actually modified by the action; all the numeric fluents that are not directly mentioned in the effects of an MMA are assumed to persist (Frame Axiom).

Therefore, within our CSP model we have to add constraints modeling the transition of each numeric fluent from a snapshot i to a snapshot i+1, taking into account whether the numeric fluent is mentioned within the effects of the i-th MMA action or not.

- **Affected Numeric Fluents:** for each i-th action in the plan, the selection of the mod_i variable binds the i-th snapshot with the $i+1$ one. More

[4] In the implementation setup, we noticed that the CSP Solver performance can be enhanced when dealing with homogeneous kinds of numeric expression. For this reason Section 5.4 reports a technique for producing a normalized form of numeric expression

precisely the constraints to be added are:

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the E $\in numEff(mod_i)$ **do**
    CSPExp := $Subst(E.exp, X, V^i)$
    CSP.addConstraint($(mod_i = k) \rightarrow (V^{i+1}_{E.nfluent}, E.op, CSPExp)$)

Here the function *Subst* creates a CSP representation of the arithmetical expression causing the numeric transition.

- **Not Affected Numeric Fluents:**

for $i = 0 \rightarrow |\pi - 1|$ **do**
  forall the fluent $\notin affected(mod_i)$ **do**
    CSP.addConstraint($V^i_{fluent} = V^{i+1}_{fluent}$)

For the fluent not affected by the action, it suffices to constrain the i-th snapshot with $(i + 1)$-th one.

Small Example

To facilitate the comprehension we will introduce a small example. Let us imagine to be in the position of building the constraints binding the effects of the *fly* action of the *ZenoTravel*[5] in the modality *cruise*.

The CSP representation will contain 4 numeric variables, i.e. $\{V_{t\_f\_u}, V_{fuel}, V_{t\_s}, V_{capacity}\}$, moreover let us assume that *distance*, *consumption* and the *speed* are modeled as constant[6].

By recalling the model of the action reported in Figure 3.5, we know that the action affects *total-fuel-used (t_f_u)*, *fuel* and *time-spent (t_s)* leaving persistent the others numeric fluents of the domain.

Here the constraints resulting for the i-th fly(cruise) action which is the one that models the plane moving from A to B:

$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{fuel} = V^i_{fuel}$ - *distance(A,B)* * avg_cruise_cons)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{t\_f\_u} = V^i_{t\_f\_u}$ + *distance(A,B)* * avg_cruise_cons)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{t\_s} = V^i_{t\_s}$ + *distance(A,B)* * avg_cruise_speed)
$(mod_{fly} = cruise) \rightarrow (V^{i+1}_{capacity} = V^i_{capacity})$

It is important to observe that while the first three constraints involve examples of affected variables, the last constraint did not. Meaning that the capacity will remain unchanged after the fly action.

[5] For the complete domain definition, see Appendix C.2

[6] In principle the constants are not explicit in our model. A technique to automatically extract them by reasoning on the problem and the plan is reported in Section 5.4

5.4 Improving the CSP Representation

In this Section we show some expedients used to generate a cheaper and optimized version of the CSP model. The improvements are obtained by reasoning on the structure characterizing the DMAP. The objectives are:

- restricting the set of numeric variables to consider
- standardizing the form of the constraints (in particular the involved numeric expressions)

5.4.1 Invariant Analysis

The major impact in the number of variables involved is given by the numeric part of the world state domain, i.e. $|X|$. In fact, the modeling of real world domains could require a big amount of numeric information. [7]

For instance, the *Planetary Rover* domain requires to model the distance among cities by means of numeric fluents "distance(a -site,b -site)". This information have to be translated within the CSP Model since each condition for the applicability of the modality for the drive action depends on the value of the distance between the two sites (see the action model for the *Planetary Rover* domain in appendix). Morover, (i) the number of connections among sites described in the state of the system may be quite big and (ii) the state variables have to be replicated as many times as there are actions in the plan. Thus the management of the CSP may become a big issue. For instance, a problem involving 100 connections among sites and a length of the plan equal to 100 actions would require 10000 CSP variables.

Therefore, by observing the transformation described in the previous section, it is quite evident that the number of the variables to handle within the CSP could rapidly increase.

By analyzing in more depth the DMAP problem, it is however worth noting that some of the numeric fluents are actually negligible. Indeed, not all the numeric fluents are subjected to change in a given plan of actions.

More precisely, given a DMAP formulation Φ, it is possible to find a set of numeric fluents which are actually constant according to Φ. More formally:

Definition 13 (Invariant Numeric Fluents)**.** *Given a DMAP* $\Phi = < \pi, i, S^i, G_{num} >$ *in a domain* $< U, F, X >$*, the set of invariant numeric fluents for* Φ *is a subset of* X *which contains only fluents not affected by action in* π_i*. That is:*
$inv(DMAP, X) = \{x | x \in X \text{ and } x \notin num\_affected(a_j) \forall i \leq j < |\pi|\}$

[7] The attentive reader can argue that the only variables actually involved in the decisional process are the action modalities variables. It is certainly correct. However: (i) the numeric information burdens the space of the information to be stored anyway, and (ii) there may be many decisional variables that, depending on the numeric information of the domain, can be actually removed if it is the case

where:
$num\_affected(a_j) = \{y | y \in X$ *and there is an assigner* $< op, f, exp > \in numEff(m) \forall m \in a_j$ *such that* $f \equiv y\}$

Since an invariant numeric fluent does not change throughout the predicted plan execution, for each numeric expression involved in Φ, it is possible to substitute each reference to x with its initial (and invariant) value.

As a consequence, some variables of our problem can be compiled away yielding a new, simplified, version for the problem Φ, namely $\Phi' = < \pi', i, S'^i, G'_{num} >$, where:

- π'_i is obtained from π_i by simplifying the numeric expression involved in the action model.
- S'^i is obtained from S^i by removing the information relative to the invariant numeric fluents
- G'_{num} is obtained from G_{num} by simplifying the numeric expression involved in the comparisons.

It is worth observing that the space of the solutions of this new formulation remains unchanged w.r.t. the original version. Indeed, no decision (the modality selection) has an impact on such numeric fluents, and hence the region of the states space which involves different assignments (w.r.t. their initial value) for such variables is actually unreachable.

It follows that the CSP formulation may take an advantage of this since fewer numeric information has to be taken into account. Indeed, given M and M' the models generated by Φ and Φ', respectively, it follows that, if Φ' is obtained as described above, we have that the number of numeric information necessary for M' is less than the ones for M. More precisely, let K be the number of invariant fluents discovered, we have that $|NUMs(M')| = |NUMs(M)| + k(|\pi| + 1)$. Finally, since the CSP size is (also) measured in terms of the number of variables involved, it is evident that a CSP translation for Φ' is more convenient than a CSP translation for Φ.

5.4.2 Effects of the Invariant Analysis

Besides the quite evident improvement in the management of the resulting CSP model, the invariant analysis produces two others major enhancements:

- some comparison can be evaluated immediately
- some non linear expression can be simplified to be linear

The Comparison Case

Let us remember that a comparison in our system has the form $< exp, \{<, \leq, =, \geq, >\}, exp'$ and that each comparison can appear in the numeric precondition of a modality and in the numeric goal definition (see Definitions 3 and 19). To show that the invariant analysis may produce some "immediate evaluations" of a comparison, it suffices to introduce just an example in which both *exp* and *exp'* become real constants. To this end, let us focus on the *Planetary Rover* domain.

Let us imagine to express the level of roughness of the terrain between location A and B by means of the numeric fluent "roughness". Moreover assume that the numeric precondition of a modality (for example of a drive action) states that the modality is executable iff the roughness of the terrain is at maximum "10.0"; then consider a plan where such a roughness is actually invariant (according to Definition 13) and equals to 13.0 given the executing plan. Therefore, the action with that modality will not ever be executable. Indeed, the propagation of the invariant numeric fluents all along the numeric expressions within DMAP will evaluate at a certain point that (roughness < 10) can be actually converted in ($13 < 10$) which is a not satisfiable condition. As an effect the system can immediately infer that such specific modality is useless and hence the domain of the action containing such a modality can be reduced.

This can happen also in comparisons mentioned in the goal definition. It may be the case, in fact, that the immediate evaluation of an expression within the goal definition may yield the (one step) proof that the DMAP is actually unsolvable.

Expressions Simplification

As thoroughly discussed in [50], as studied in [47], and similarly as others planning systems ([45] [23]), the numeric expressions that can be handled by our system are limited to the linear case. That is, a given expression cannot contain (a.mong sub-expressions) arithmetical operations of the form $\{exp, \{*, \setminus\}, exp'\}$, where both *exp* and exp' involve at least a numeric fluent. Given a model of the action containing non linear expression, it may be the case that, via invariant analysis, some expression can be simplified to the linear case.

Similarly to the comparison case, to show the effectiveness of the invariant analysis it is sufficient to have just an example of "linearization". Hence, let us consider the *Planetary Rover* domain.

Let us recall the model of the drive action and let us focus on the numeric effects on the time via a given modality. Namely:

`increase(time,(\ (distance A B) (speed r1) ))`

`(distance A B)` and `(speed r1)` are of course modeled as numeric fluents in our model of action and for this reason the arithmetical expression is not

linear. Indeed, in a linear expression, the two numeric fluents are not allowed to be combined by means of either a multiplication or a division.

However, no action in such a domain includes effects which changes directly the *speed* of the rover, neither the *distance* between two cities. For this reason both *distance* and *speed* can be substituted with their specific initial values, producing an effect which just depends on the previous value of the variable time summed to a constant factor, namely the product of the initial value of the *distance(a,b)* and the *speed* of the rover.

5.4.3 Normalizing Expressions

This subsection explains the process to translate the numeric expressions involved in DMAP in a normalized form. The technique generates a compact and *flat* representation for the numeric expressions in the CSP model too. Experiments performed with this reformulation turns out in a (quite big) speeding up of the CSP resolution task [8].

More precisely, a *Normalized Numeric Expression* is defined formally as:

Definition 14 (Normalized Numeric Expression, adapted from [50])**.** *Given a world state domain $< U, F, X >$, a normalized numeric expression on X has the form of a summation $\sum_{i=1}^{j=|X|} a_i f_i + a_0$, where $a_k \in \Re\ \forall 0 \leq k \leq |X|$ and $f_i \in X$*

The following algorithm shows the pseudo code for the generation of a normalized expression starting from a numeric expression as the ones given in input to our system.

For convenience, the reported code in algorithm 1 assumes that the normalized expression is organized as a vector of real values representing the coefficients of the summations. In position "0" there is the value assigned to the constant a_0, while in the rest there are the coefficients of each numeric fluent of the problem. Of course we implemented the expression as a linked list for efficiency reasons.

The algorithm distinguishes three cases. The first two cases represent the base-case of our recursion, where the expression is identified as either a real constant or a numeric fluent. So, the algorithm instantiates the real number (or the numeric fluent coefficient) within the respective vector position. That is it puts the real value in position "0" and 1.0 in the numeric fluent address.

In the third case, i.e. whenever the expression is actually a binary operation, the recursive step firstly normalizes both the left and the right part of

[8] In our experiments we employed the Choco solver (`http://www.emn.fr/z-info/choco-solver/`) which implements the state of the art algorithms for the CSP resolution and supports the recent standardization for the constraint programming in Java (`http://www.jcp.org/en/jsr/detail?id=331`). As stated in the documentation, it is desirable to not over-stress the csp expressions mechanism due to the impossibility of using customized propagation techniques. In particular by this reformulation of the expression we employ the meta-variable "sum" which is a well supported kind of expression in Choco.

Algorithm 1: Normalize Expression

```
   Input: expr - expression
   Output: norm_expression
 1 norm_expression expr' ;
 2 if expr is a realconstant then
 3 |   expr'[0] := expr ;
 4 |   return expr'
 5 else if expr is a numeric fluent then
 6 |   norm_expression expr' ;
 7 |   expr'[position(expr)] := 1.0 ;
 8 else
 9 |   norm_expression left = Normalize(left(expr));
10 |   norm_expression right = Normalize(right(expr));
11 |   expr' := combine (left,right,op(expr));
12 return expr'
```

the expression; then the *combine* procedure performs arithmetical operations to merge the two normalized expressions in one according to the operator defined in *op(expr)*. Let us remember that since our numeric expressions are restricted to be linear, the *combine* procedure does not cope with multiplication and division between two expressions which contains at least a numeric fluent. The normalization task has to performed *after* the invariant analysis. This guarantees to normalize also those expressions which are in principle non linear, but that have been simplified to be linear (see previous Section).

It is important to note also that, to do not encounter non linear numeric expressions, the normalization has to be performed just after the invariant analysis.

A similar normalization has been defined in [50]. However our purpose is different as we do not exploit the normal form for solving a planning task. Differently the normalized form provide us a means for an efficient translation toward the CSP constraints.

In fact, following the definition reported in 3, a numeric expression is recursively defined in terms of either a real constant, a numeric fluent or a binary operation ($\{+, -, *, \backslash\}$) between two expressions. Hence, the original form of numeric expressions is a tree. A naive translation mechanism as the one presented in Section 5.3, will translate the CSP constraint as a tree too, and this can become an issue from the CSP solver point of view.

5.5 Discussions and Further Improvements

This Chapter reports the mechanism for transforming a DMAP in a Constraint Satisfaction Problem. In particular, we have introduced the fundamental blocks

to transform a generic DMAP in a set of variables and constraints to be given as input to a generic CSP solver. The solution of the corresponding CSP will be a new allocation of modalities for the DMAP.

The mechanism inherits the fundamental steps performed by [59] and [36], but extends the basic conversion for handling numeric expressions in the context of MMAs.

In addition, the Chapter have introduced some improvement that can be done for generating a cheaper CSP (in the size of variables to be considered), and which (the CSP) encompasses only a homogeneous set of constraints (i.e. involving normalized numeric expressions). This new formulation is key to scale up the performance of our system as the DMAP can focus only on the numeric information which are actually interesting for the problem at hand. Moreover, thanks to a unified vision for the constraint, the CSP solver can employ specific propagation techniques.

The invariant analysis is one of the most common features in recent automated planners and is used as a mechanism to efficiently focusing the planner just on a portion of the state space. The process has been applied to both the propositional and the numeric setting. In those contexts, many invariant numeric fluents can be inferred by considering the domain and the problem, for which they are "plan" independent.

In our approach, we can retrieve more invariant numeric fluents and, besides those that are not interesting for the problem, we can also remove the ones that are not relevant for the current plan of actions.

We see two main enhancements that can be applied to our approach. The former refers on the reasoning that can be performed on the nature of the numeric fluents involved when such fluents actually represent resources. Indeed, while some resources can be renewable, other ones can be just consumable and the CSP has to be informed about that, meaning that other constraints can be added. It is well known also that as long as the number of constraints increases, the domain for the variable tends to decrease as well, favoring, unquestionably, the CSP resolution task.

The latter is a more powerful reachability analysis. Similarly to [50], [45] and to the more general contribution of the *graphplan* structure ([1]), one can reduce the search space by removing all the states which can be proved (via *graphplan* relaxation) to be not reachable with the modalities at hand.

Chapter 6

FLEX-RR :: Supervising Multi Modality Plans

This Chapter presents the *FLexible EXecution via Reconfiguration and Replanning* (FLEX-RR) system. FLEX-RR is a system for handling the execution and recovery, if needed, of the multi modality plan at hand. The FLEX-RR architecture relies on the notions and the tools introduced and described in the previous Chapters. In particular the notion of the Multi Modality Plans presented in Chapter 4 and the CSP mechanism developed in Chapter 5.

6.1 Introduction

In the previous Chapters we have formally defined the notion of a Dynamic Modality Assignment Problem (DMAP), which grounds on the concept of Multi Modality Actions (MMA), and we have studied the relation among the DMAP and the more general Multi Modality Planning Problem (MMPP). We have also seen a method for transforming the declarative approach of a DMAP into a Constraint Satisfaction Problem, that gives us a computational model for the actual management and resolution of the DMAP, while the MMPP can be solved by a general purpose PDDL planner.

The methodology presented in this Chapter integrate the techniques for a real system pursuing the main objective of this thesis, that is, the robust execution of plans for domains characterized by the presence of (continuous) numeric information (e.g. consumable resources). To this end, in particular, the methodology we are going to describe deals with the supervision of Multi Modality Plans for environments not completely predictable, where unexpected changes in the state may threat the plan feasibility at any instant of time.

The proposed system, namely FLEX-RR, takes its name from its main objec-

tive, namely "flexible execution". The flexible execution is intended as a means for the robust execution of the plan in tolerating deviations from the nominal expected behavior. The FLEX-RR suffix (i.e. RR) stands for Reconfiguration and Replanning, which are the adopted mechanisms for achieving flexibility all along the plan execution. In particular we have that:

1. the *Reconfiguration* is the feature offered by the system for the dynamic adaptation of the actions modalities. Substantially, a *Reconfiguration* problem is a particular instance of the DMAP which can be encountered during the execution;

2. the *Replanning* is the feature allowing the system to replace the old (presumably unfeasible) plan with a new course of actions. It corresponds to the resolution of a MMPP.

FLEX-RR aims at being an on-line tool for the intelligent supervision of the plan. To this end, it shall act and decide as fast as possible, while maintaining a certain degree of flexibility to unexpected contingencies. In few words, the system to be effective has to show an high degree of competence, that is FLEX-RR should guarantee flexibility with a limited computational effort. The double level of control provided by the *Reconfiguration* and *Replanning* has been designed with this objective in mind.

To be independent from the particular domain of application, FLEX-RR employs the multi modality action formalism which, as we have seen in Chapter 3, is a slight extension of the PDDL action model. As PDDL, the adopted language does not make any assumption on the specific scenario of interest, rather it is possible to describe the agent behavior in a declarative way; for this reason the FLEX-RR mechanism is supposed to be quite general in the context of robust plan execution.

FLEX-RR is similar in some aspects to the continual planning paradigm as the ones reported in [15, 34] but it specializes this general approach for agents dealing with plans involving numeric information, e.g. consumable resources.

The Chapter firstly proposes the architecture of FLEX-RR (Section 6.2), then an algorithmic formulation for the main control loop implemented by the agent (Section 6.3). Section 6.4 explains how FLEX-RR manages the CSP model, while Section 6.5 describes the reconfiguration mechanism. Finally, Section 6.7 shows a run example of FLEX-RR.

6.2 Architecture of the system

As anticipated in the previous Section, FLEX-RR can be seen as a particular instance of a continual planning agent.

The continual planning is a paradigm for the implementation of agents devoted to perform tasks in the real world environments, where a continual revision of the plan is necessary in order to account the multiple and unpredictable contingencies that may occur[1]. In order to successfully reach the objectives, a continual planning agent is allowed to interleave planning and execution all along the task assigned. Precisely, the agent is supposed to have two main capabilities. That is, on one hand, the agent should be able to understand when interrupting or revising the execution (because for example, the plan is not valid anymore). On the other hand, the agent should be able to recover from the impasse. Of course, both steps have to be executed in a timely fashion to be useful.

Analogously to continual planning techniques, FLEX-RR leads the execution by assuring that the plan remains *valid* all along the task. FLEX-RR adopts a *conservative* behavior, since each action is performed only if the *whole* plan is consistent with the goal. Indeed, while in principle an action of the plan could turn out to be executable in a given state of the system, FLEX-RR prefers to interrupt the plan execution whenever the whole plan consistency is compromised. In this case, differently from the standard continual planning approach, but similarly to plan-adaptation based mechanism, FLEX-RR anticipates the replanning from scratch with a more focused mechanism based on a reconfiguration of modalities. The FLEX-RR task ends when either a plan is completed with success or the set of objectives is considered unreachable.

FLEX-RR exploits several components for its supervision task. For this reason, the next Section briefly introduces which are the fundamental blocks and how they interacts among each other.

6.2.1 FLEX-RR Architecture

Figure 6.1 summarizes the architecture at the base of the strategy implemented by FLEX-RR.

From the top and the bottom of the Figure it is possible to understand how FLEX-RR interacts with the environment. In particular, from the top, FLEX-RR receives the plan to be executed (in form of a MMAs) and the set of goals to reach and produces an outcome which indicates whether the task has been achieved or not. Depending on the particular domain of application, the plan can be generated either by an automated or a mixed initiative planner (e.g., [20],[16]). It is important to note that the initial plan is a solution for a MMPP[2].

[1] Let us remember that an alternative approach to the continual revision consists in anticipating the contingencies at planning time. However, this kind of strategies only apply in contexts where the number of contingencies is limited to few cases

[2] This assumption can be relaxed as well; indeed FLEX-RR has all the facilities of repairing a plan which can be hence flawed from the very beginning of the task

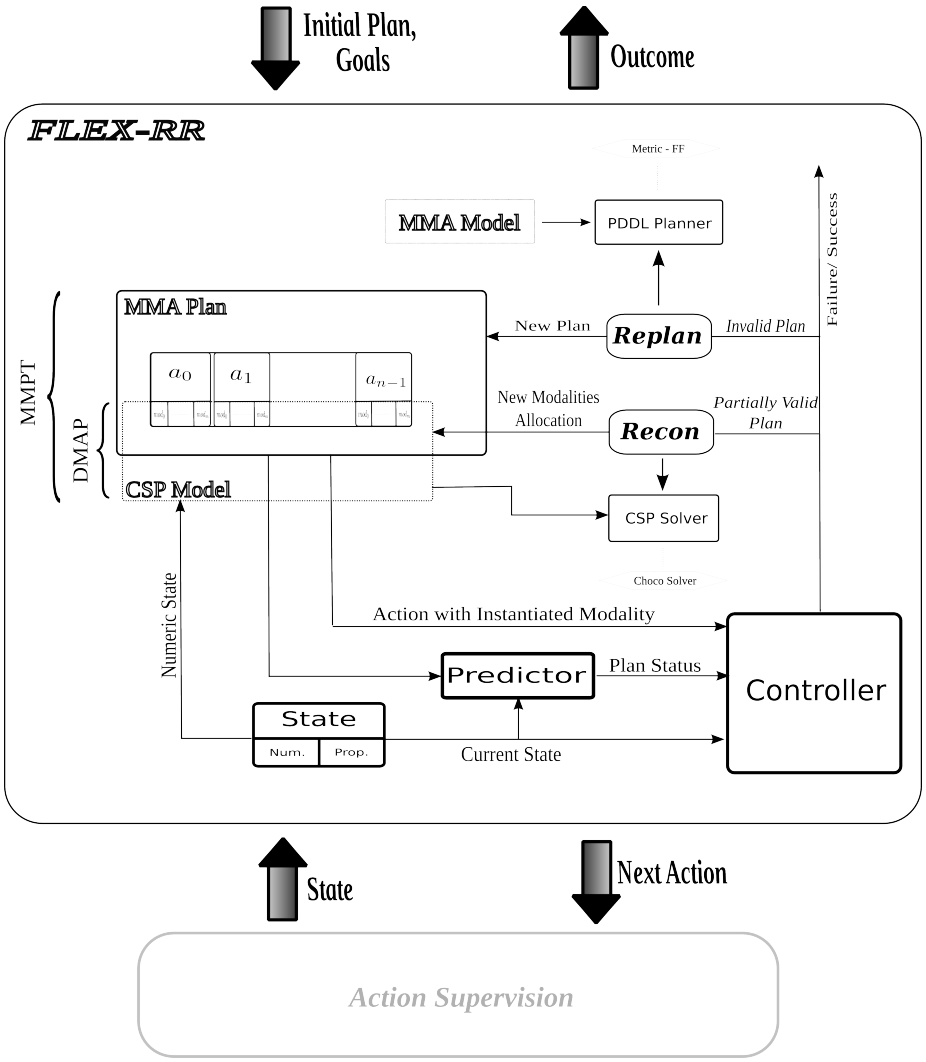


Figure 6.1: Internal FLEX-RR architecture

At the bottom of the Figure 6.1, there is the functioning of FLEX-RR for the on-line phase. In particular, FLEX-RR submits actions to lower level of control (e.g. Action Supervision, see Chapter 9) while updating the internal state upon the information coming from the outside (e.g. the sensors or an abstraction of them).

The main structures and modules necessary for an effective execution and adaptation process are the following:

- the *MMA plan*, which is the total ordered set of actions to be executed. As we have seen in Chapters 3 and 4, each action is equipped with a set of execution modalities, according to the MMA Model; initially the MMA

plan is the input of the FLEX-RR system;

- the *CSPModel*, namely the structure introduced in Chapter 5, which encodes the relation between the modalities and the numeric fluents of the problem;
- the current *state* of the system, which handles both the numeric and the propositional information representing the current situation;
- the *Predictor*, which assesses the plan feasibility according to the upcoming situation;
- the *Controller* module, which decides the next action to execute. In particular it can repair the plan whether it is not valid or emit a failure when the goal is not reachable anymore.
- on the top, next to the plan representation, the two possible recovery mechanisms that can be applied: *ReCon*, which exploits a CSP solver, and *Replan*, which exploits a numeric PDDL planner. The former interacts with the CSP Model above, while the latter with the MMA model which contains the actions available by the planner[3].

In the current implemented architecture (as reported in Figure 6.1), the employed planner is the Metric-FF planning system ([50]) whereas the Choco library (`http://www.emn.fr/z-info/choco-solver/`) has been used as CSP solver.

The main contribution of FLEX-RR is the way the unexpected contingencies are handled. In this perspective, as a main difference w.r.t. a standard continual planning system, FLEX-RR distinguishes between three different plan states; that is, a plan can be recognized to be either *valid* or *partially valid* or *invalid* (see Definition 10). As anticipated, the execution proceeds only when the plan turns out to be valid, while both the invalid and the partially valid status impose to FLEX-RR to suspend the actions submission until the the plan recovers from the impasse.

In the architecture, the plan status is computed by the predictor, while the *Controller* module is the one in charge of executing the action or deciding the interruption of the plan execution. The two types of inconsistencies (*partially valid* and *invalid*) allows the *Controller* module to employ two different levels of recovery over the plan execution, respectively. The former, applied when the plan is considered partially valid, and corresponds to the reconfiguration mechanism; that is, an invocation of the CSP solver for the resolution of a specific instance of DMAP. The latter corresponds to solving a new MMPP, i.e. replanning.

[3]In this context the MMAs are represented by means of schema, similarly to PDDL.

In the next Section we show an algorithmic formulation of the overall control loop, that is, from the point of view of the *Controller* module.

6.3 FLEX-RR: Main loop

This Section shows the main control loop of FLEX-RR, which is clear since the organization of the components of the system (see Figure 6.1). The continual planning process circularly follows the flow of the information which starts from the plan, enters in the *Controller* module and returns on the top (and hence in the plan) when the execution has been interrupted (because of inconsistencies or the plan is terminated). The plan is the list of actions to perform and the control over the plan constantly updates such a list for taking into account the actual situation of the environment or simply for removing the action already performed.

Algorithm 2: FLEX-RR

```
   Input: I, G, π
   Output: Success or Failure
 1 CSPModel=build-CSPModel( I, G, π)
 2 i=0
 3 S=I
 4 while i < |π| do
 5     senseAndUpdate(i,S,CSPModel)
 6     plan-status = propagate(S, G,π)
 7     if plan-status is valid then
 8         a_i = getActionAt(π, i)
 9         execute(a_i)
10         i++
11     else if plan-status is invalid then
12         π = Replan(S,G)
13         CSPModel = buildCSPModel(S, G, π)
14         i = 0
15     else if plan-status is partially-valid then
16         π = ReCon(CSPModel, i, π)
17         if π = ∅ then
18             π = Replan(S,G)
19             CSPModel = buildCSPModel(S, G, π)
20             i = 0
21 if S ⊢ G then
22     return Success
23 else
24     return Failure
```

The algorithm reported in 2 starts by initializing two important structures:

the $CSPModel$ and the state S. The state S corresponds to the initial state I of the problem; this structure will be updated during the plan execution by the acquisition of information from the environment. The $CSPModel$ is initially built by considering the MMPP Π at hand. In particular the $CSPModel$ focuses on the numeric aspects of the MMPP, thus the propositional information are not managed (see Chapter 5 for further details).

As we will see, during the execution of the plan π, solving the problem Π, the $CSPModel$ will be modified by adding and deleting constraints, or by asserting new information coming from the environment. This guarantees that the $CSPModel$ can be promptly employed by the $CSPSolver$ whenever it is necessary (i.e. for a reconfiguration).

The algorithm iterates over the plan actions as long as there is at least one action to execute. The algorithm returns either *Success* or *Failure* depending on whether the status S reached after the execution of π satisfies or not the goal G. The iteration may be also interrupted when the replan mechanism is not able to find a solution. In that case the replan returns a plan of size 0 and the algorithm returns with a *Failure*[4].

At each iteration, the algorithm observes and updates the world state S (the *senseAndUpdate* function). Note that also the $CSPModel$ structure is updated at this step with new observed information; in this way, the CSP representation includes all the relevant pieces of information for solving a new DMAP whenever it arises. After the observations gathering, the algorithm assesses the state of the plan π, i.e. it evaluates whether the plan is still *valid* or not. To accomplish this step, the function *propagate* is invoked to estimate the impact of the updated state S into the planning problem Π. Intuitively, the propagation verifies whether the goal G can be achieved from state S by simulating the execution of all the remaining actions in π_i without any change in their modalities. In particular the simulation returns a prediction of the final state S' which can be matched with the goal G to check if G is satisfied or not in S'. In the positive case, the plan is *valid* and its next action a_i is selected for the execution with the modality it is currently associated with[5]. Otherwise, the plan is either *invalid* or *partially-valid*. With reference to the architecture reported in Figure 6.1, the propagation task is performed by the Predictor module[6].

In case the plan is *invalid*, the *Replan* module is invoked to build a new plan from the current state S to the goal state G[7]. Note that, when a new

[4]However, the algorithm does not exclude the case in which unexpectedly the agent is in a state where the goal is actually satisfied. Also this case will be intercepted so that the algorithm can return the correct outcome, i.e. *Success*

[5]Actually, the same CSP solver can be used to assess the validity of the plan. This can be achieved by simply restricting the choice of the modalities to the previously instantiated modalities.

[6]In Chapter 8 we will see an extension of such mechanism which allows to focus only on thos information which are actually *relevant* for the problem at hand

[7]To allow the interaction between our software module with a generic PDDL numeric plan-

Algorithm 3: SenseAndUpdate

Input: i, S, $CSPModel$
Output: updated S and $CSPModel$
Obs = <SenseWorld>
$S = updateStatus(S, Obs)$
if $i = 0$ **then**
  addConstraint($CSPModel, V^0 = S$)
else
  addConstraint($CSPModel$, mod_i=$exec$)
  addConstraint($CSPModel, (mod_i = exec) \rightarrow (V^{i+1} = S)$)

plan π is returned, this plan substitutes the old one, so the execution restarts from the first action of the new π. For this reason, both the counter i and the $CSPModel$ need to be re-initialized.

In case the plan is *partially valid*, we have detected a DMAP problem, and FLEX-RR tries to solve it by invoking the reconfiguration module (ReCon). ReCon finds (if exists) an alternative assignment of the modalities to the actions still to be performed. Note that, since the DMAP problem might not have solutions, ReCon could return an empty plan; thus, also in this case, the algorithm will invoke a replanner to resolve the impasse.

The next two algorithms explain in detail: (i) how the CSP model is updated along the plan execution, (ii) how the CSP solver is invoked, and (iii) how the PDDL planner is invoked.

6.4 Updating the state and the CSP model

Algorithm 3, *SenseAndUpdate*, takes in input the index i of the last performed action, the current state S of the world, and the CSP model to update.

First of all, new observations Obs, collected from the environment, are used to update the state S; that is, both the propositional and the numeric state. Then, the algorithm updates the CSP model depending on the fact that it is the first step (i.e., i equals 0) or a subsequent one. At the first step of execution, the CSP model is updated by imposing that each numeric variable in V^0 assumes the value of the corresponding numeric fluent in S. Where V^0 is the subset of numeric variables within the CSP model associated with the level *0*. This allows FLEX-RR to deal with an initial world state that is different from the assumed one.

In any other execution step (i.e., $i > 0$), the CSP model is modified by

ning (Metric-FF, [50] in our case), we apply the conversion mechanisms sketched in Chapter 4 and reported in Section 6.6. When the planner is invoked, each MMA model is flattened to the traditional PDDL model; these models are therefore used by the planner. Then, once the plan is computed, the PDDL actions are newly transformed into MMAs.

changing the modality of the last performed action, a_i, to *exec*. This special modality has two important roles. First, when an action has modality *exec*, it cannot be considered during the resolution of a given DMAP as its modality cannot be changed any more. Second, when action a_i has modality *exec*, the variables in V^{i+1} are no longer constrained. The constraints are in fact defined according to the modalities specified in the MMA model of a_i, and *exec* is not among them. This allows us to assert within $CSPModel$ any observations coming from the real world even though they are completely unexpected (i.e., not foreseen by any modality associated with a_i). Note that in this way we do not need to re-build the CSP model from scratch, but simply adjust the same model progressively at each execution step. When a new DMAP occurs, the CSP model already encodes all the information required for the DMAP resolution.

6.5 Reconfiguring the plan with ReCon

Algorithm 4 shows the high-level steps of the *ReCon* module. In particular, ReCon has to solve a DMAP problem, and takes in input the plan π to be repaired and the CSP model which, as said above, already encodes all the pieces of information relevant for the solution of the DMAP. The CSP knows the portion of the plan that has to be reconfigured, as this corresponds to the set of actions not yet performed, which are the only decisional variables to be used during the resolution. In fact the CSP encodes the modality-variables for the executed action by exploiting the special modality *exec*. Moreover, the CSP is provided with the earlier observation thanks to the constraint binding the *exec* modality with the information acquired from the environment (see previous Section).

ReCon tries to solve a DMAP by means of a CSP solver. If the CSP-solver finds a solution, this is extracted and used to reconfigure the plan π which is therefore returned to FLEX-RR. In other words, each action a_i still to be performed is assigned the modality selected by the CSP-solver (see function *reconfigurePlan* in Algorithm 4). On the other hand, when the CSP-solver does not find any solution, ReCon returns an empty plan.

The *reconfigurePlan* iterates over the (*non-exec*) modality variables of the CSP and allocates the assignments generated by the solver to the actions of the plan.

6.6 Replanning

This Section describes the task accomplished by the *Replan* module by means of the algorithm 5. The main idea of the procedure grounds on the fact that

Algorithm 4: ReCon

Input: π, $CSPModel$, G_{num}
Output: reconfigured π or $\emptyset$
$Solution$ = CSP-solver($CSPModel$)
if $Solution = null$ **then**
  return $\emptyset$
else
  reconfigurePlan($Solution$, π)
  return π

Algorithm 5: Replan

Input: S - state, G - goal
Output: π - MMAPlan or $\emptyset$
1 $Map < PDDLAction, exec\_modality >$ mod_of = {};
2 $PDDLModel$ A = {};
3 **foreach** *mma* $a \in MMAModel$ **do**
4 **foreach** *exec_ modality* $m \in mod(a)$ **do**
5 PDDLAction a' = flatten(a,m);
6 mod_of = mod_of $\cup$ {a,m};
7 A = A $\cup$ {a'};
8 $PDDLPlan$ ω = solve(A,S,G);
9 **while** $\omega \neq \emptyset$ **do**
10 PDDLAction a"=head_remove(ω);
11 MMA a"' = abstraction(a");
12 inst_mod(a"') = mod_of(a");
13 append(π,a"');
14 **return** π

each MMA a can be flattened in a set of PDDL 2.1 actions $\{a'_0, a'_1, ..., a'_{m-1}\}$ where m is the number of modalities of a and the model of the i-th PDDL action is created by means of:

- the propositional precondition of a
- the numeric precondition and the numeric effects of the i-th modality of a, i.e. $mod(a)_i$

The procedure takes in input a state of the system and the goal conditions, and returns either the solution of the planning problem (MMPP) or an empty plan. As a first step, the algorithm instantiates two structures: mod_of, which is a mapping between a PDDLAction and an execution modality alias, and A, which is the PDDLModel of the actions to be considered by the planner.

The algorithm iterates over all the modalities for each MMA present in the MMAModel[8]. For each step of the iteration the process creates a PDDLAction

[8]Let us note that here we consider action templates

which encompasses both the propositional information of the action and the numeric information present in that modality. To keep trace of the relation between the PDDLAction and the modality it represents, this first iteration stores the mapping in the pair mod_of.

Having created the set of $PDDLActions$, the procedure can hence invoke the PDDLPlanner (e.g. $MetricFF$), see line 8, and wait for the computation. Once the computation is finished the obtained solution (if any), will be reorganized in an MMA plan. More precisely we iterate over the PDDL plan and we *abstract* each action to obtain newly an MMA. To point out which is the instantiated modality for the abstracted MMA, we exploit mod_of as anticipated before (see line 12).

6.7 An example of how FLEX-RR intervenes

To exemplify how FLEX-RR is able to monitor the execution of a plan and to repair it in case the plan is no longer valid, let us consider two examples in the *Planetary-Rover* and in the *ZenoTravel* domain, respectively.

6.7.1 Planetary-Rover Example

Let us continue on the example reported in Chapter 4, where Figure 6.2 reports both the initial problem and the resulting multi modality plan[9].

As a first step FLEX-RR is up to build the $CSPModel$. To this end it computes all the variables and the constraints requested for representing the 6 different time snapshots of the state transitions caused by each action of the plan to be executed. As reported in Chapter 5, each snapshot will contain the value of each numeric information of the domain (e.g. the power at time 0, the time spent at time 1 and so forth).

Analogously to the example reported in Section 4, and moreover according to (i) the model of the actions, (ii) the plan, and (iii) the initial state of the system, FLEX-RR makes the prediction reported in Figure 6.3.

As reported in Algorithm 2, at each step of the execution, FLEX-RR substitutes the snapshot of the variables values for that step, with the current observation.

Thus, the *Predictor* module can capture the possible discrepancies that can be encounter all along the plan execution.

[9]For readability reasons, the problem reported simplifies a part of the numeric information which instead are completely specified in Appendix C

```
/*INITIAL STATE*/

/*propositional info*/
(at (r1,A))
(road (A,B))
(road (B,C))
(road (C,D))
(road (A,D))
/*numeric info*/
(= (distance(A,B)) 15)
(= (distance(B,C)) 35)
(= (distance(C,D)) 45)
(= (distance(A,D)) 10)
(= (power r1) 1000)
(= (memory r1) 4)
(= (time) 0)

/*GOAL CONDITIONS*/

/*proposition goal*/
(info_acquired(r1,B))
(info_acquired(r1,C))
/*numeric goal*/
(> (power r1) 300)
(< (time) 50)
(= (memory r1) 4)
```

```
/*MMA PLAN*/

1: Drive(A,B)(cruise)
2: TP(B)(HR)
3: Drive(B,C)(cruise)
4: TP(C)(HR)
5: Comm(C)(CH1)
```

Figure 6.2: Planetary Rover Domain :: Problem and MMA Plan

```
PREDICTION

/*propositional information*/
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric information*/
(= (power r1) 334)
(= (time) 47)
(= (memory r1) 4)
```

Figure 6.3: Planetary Rover Domain :: Step 0 Prediction

```
PREDICTION

/*proposition goal*/
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric goal*/
(= (power r1) 290)
(= (time) 47)
```

Figure 6.4: Planetary Rover Domain :: Step 2 Prediction

Scenario 1

Let us assume that, at time 0 there are no discrepancies w.r.t. the assumed initial state reported in Figure 6.2 and after the first *drive* action, the system notices an unforeseen consumption of the power. The information is stored in the current snapshot and, as a consequence, the predictor estimates the power consumption at the end of the mission to be different w.r.t. what has been predicted at time 0; see Figure 6.4 for the new computed prediction.

By matching the prediction and the goal conditions it is quite clear that the plan turns out to be *partially valid*. Indeed, while the propositional conditions continue to be satisfied, the numeric conditions do not (i.e., $290 > 300$ does not hold). Therefore the *Controller* module invokes the CSP solver to verify if it is feasible to accommodate the current plan with a different configurations of action modalities.

In particular, in the current situation, the actions which could reestablish the validity conditions are the *drive* and the *comm* action, since they are the only conditions where the power of the robot depends on the selection of the modality (in our planetary domain we assumed that the take picture power consumption is negligible).

The solution found by the CSPSolver is reported in Figure 6.5.

It is worth noting that another solution could be to change the modality of the third drive action to be "safe" instead of "cruise". However, while the power would be reestablished, the time would not.

With the new configuration, the final prediction of the state turns out to be consistent with the goal and the execution can proceed. See Figure 6.5.

Scenario 2

Assume that in a new situation, till the second step there was no discrepancy from the expected state so that the rover is not requested to perform any form of repair of the current plan.

However, suppose that, at the third step of the execution, before performing

```
/*MMA PLAN*/

1: Drive(A,B)(EXEC)
2: TP(B)(HR)
3: Drive(B,C)(cruise)
4: TP(C)(HR)
5: Comm(C)(CH2)
```

```
PREDICTION

/*propositional information*/
...
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric information*/
(= (power r1) 310)
(= (time) 45)
(= (memory r1) 4)
```

Figure 6.5: Planetary Rover Domain :: Situation 1, Reconfigured Plan and New Prediction

the Drive(B,C) action, the rover estimates that the data produced and stored within the memory is actually more than expected. Since the initial budget is quite limited, the memory turns out to be completely insufficient for the achievement of the next TP. The predictor indeed estimates that the final state is actually unreachable (and hence undefined, see Definition 6) due at a missing precondition for an action within the plan. Indeed the model of the TP imposes that the action, to be applicable, requires that at least one unity of memory is available (i.e., the LR modality).

Even if the plan turns out to be just partially invalid, the CSPSolver cannot find an assignment of modalities, for this reason ReCon return with an empty solution.

The only possibility is to find an alternative course of actions for the problem at hand. To this end the replanning mechanism is invoked.

Figure 6.7.1 shows the result of the replanning and the new final state prediction. It is worth noting that it has sufficed to add a further communication action to be accomplished as soon as possible.

It should be also noticed that this has been possible thanks to the absence of constraint on the overall cost for the communication. If the problem had contained a constraint in this variable, also the replanner wouldn't have found a solution.

Two similar scenarios can be considered also for the *ZenoTravel* domain.

6.7.2 ZenoTravel Domain Example

Our problem P involves three people (`P1`,`P2` and `P3`), four airports (`A0`, `A1`, `A2`, `A3`) and one airplane `F1`. The configuration of the main information of the initial state[10], the goal statement and the plan for this problem is reported in Figure

[10]It is important to note that other propositional information are important to understand the applicability of the action. For instance the connections among the airports

```
/*MMA PLAN*/

1: Drive(A,B) (EXEC)
2: TP(B) (EXEC)
----replan----
1: Comm(C)(CH1)
2: Drive(B,C)(cruise)
3: TP(C)(HR)
4: Comm(C)(CH1)
```

```
/*propositional information*/
...
(info_acquired(r1,B))
(info_acquired(r1,C))

/*numeric information*/
(= (power r1) 334)
(= (time) 47)
(= (memory r1) 4)
```

Figure 6.6: Situation 2 :: Replan and New Prediction

6.7.

Let us suppose that the execution of the first two actions in the plan does not raise any problem. However, after the execution of action `fly_F1_A1_A2(cruise)` (index `2`), the flight `F1` has reached airport `A2`, but it has consumed a greater amount of fuel and time than expected. As a consequence, the resulting execution state *S3* is different from the predicted one.

ReCon has to solve a new DMAP by finding a new assignment of modalities to the actions not yet executed. For instance, it finds the reconfigured plan of Figure 6.8.

It is easy to see that the modality of `fly_F1_A2_A3` has been changed from zoom to cruise in order to reduce the fuel consumption, so that the constraint on fuel is no more violated (see e.g., the MMA model in Figure 3.5). However, to compensate the delay caused by such a change, also the modality of the actions `debark_P1_F1` and `debark_P3_F1` have been changed from *normal* to *express*, so that also the constraint on time is satisfied. The actions marked as *exec* are not addressed by ReCon.

After these changes, the plan is again *valid*, and its execution can resume from the current state S3.

It is worth noting that the new allocation of modalities produces a new plan π' which is very close to the original plan π. In Section 7, we will discuss the importance of keeping a repaired plan as stable as possible (i.e., as close to the original plan as possible), and how the stability can be measured.

In the previous example, FLEX-RR has been able to find a solution by means of ReCon without the need of replanning. This is not always the case. Let us suppose that the execution of `fly_F1_A1_A2` (in *zoom* modality), is affected by a very large deviation in the fuel consumption.

Also in this case the plan is *partially valid*, and therefore ReCon is invoked for solving a DAP. However, since the deviation on the fuel is significant, it is no possible to restore the validity of the plan via a simple reconfiguration. Thus, ReCon fails and FLEX-RR invokes the replanner. The new planning task consists in finding a plan that, starting from the current state S3, achieves the

```
/*INITIAL STATE*/

/*propositional info*/
(in P1 A1)
(in F1 A1)
(in P3 A2)
...

/*numeric info*/
(= (time-spent) 0)
(= (fuel) 8000)
(= (total-fuel-used) 0)
...

/*GOAL CONDITIONS*/

/*proposition goal*/
(in P2 A2)
(in P1 A3)
(in P3 A3)

/*numeric goal*/
(< (time-spent) 21000)
(< (total-fuel-used) 10000)
(> (fuel) 0)
```

```
/*MMA PLAN*/

0: board_P1_F1(normal)
1: board_P2_F1(normal)
2: fly_F1_A1_A2(cruise)
3: debark_P2_F1(normal)
4: board_P3_F1(normal)
5: fly_F1_A2_A3(zoom)
6: debark_P1_F1(normal)
7: debark_P3_F1(normal)
```

Figure 6.7: ZenoTravel Domain :: Initial Problem and Plan

```
0: board_P1_F1(exec)
1: board_P2_F1(exec)
2: fly_F1_A1_A2(exec)
3: debark_P2_F1(normal)
4: board_P3_F1(normal)
5: fly_F1_A2_A3(cruise)
6: debark_P1_F1(express)
7: debark_P3_F1(express)
```

Figure 6.8: ZenoTravel Domain :: Reconfiguration

```
0: board_P1_F1(exec)
1: board_P2_F1(exec)
2: fly_F1_A1_A2(exec)
--replan--
0: debark_P2_F1(express)
1: board_P3_F1(express)
2: refuel_F1_A2
3: fly_F1_A2_A3(zoom)
4: debark_P1_F1(express)
5: debark_P3_F1(express)
```

Figure 6.9: ZenoTravel Domain :: Replanning

same set of propositional and numeric goals. In this case the replanner finds the solution reported in Figure 6.9.

Finally, it is also worth noting that the replanner may fail, since not all the anomalous situations encountered during the execution are repairable. More important, in many real-world scenarios, the agent must react to unexpected situations in a short amount of time. That is, FLEX-RR has to find a solution within a given threshold; otherwise, the plan goals must be revised and a new plan must be synthesized.

6.8 Conclusions

This Chapter presented the FLEX-RR architecture. FLEX-RR introduces the supervision task performed for handling multi modality plans and clarifies the use of the tools presented in the previous Chapters.

FLEX-RR is practically a special instance of a continual planner. It specializes the continual planning approach for agents reasoning in tasks involving numeric information as for instance consumable resources. The main contribution of FLEX-RR is the ability of robustly performing task by adapting the current plan of action in facing unexpected contingencies that may arise during the execution. Indeed, FLEX-RR can take advantage of two different recovery steps. The first one, the reconfiguration step focuses on the action modalities of the plan. the second one, the replanner step can decide of both adding and removing action by exploiting all the state space of MMA plans. The reconfiguration mechanism is hence less flexible but more efficient, while the replanning has the maximum flexibility but less efficiency.

The FLEX-RR is designed to be *modular*. On one hand, we can employ different configurations: FLEX-R, which is the system with only the Reconfiguration facility, FLEX-REPLAN which is a system with the only Replanning facility, and FLEX-RR which contains all the functionalities presented. De-

pending on the specific domains and/or requirements, the user can opt for the desired configuration. On the other hand, since the methodology proposed does not make any particular assumption about the replanner and the solver used, FLEX-RR architecture can be upgraded without particular efforts. Therefore, the system can easily benefit from advances in the planning and CSP communities, by exploiting more efficient CSP and/or PDDL numeric planning.

Section 7 will report a thorough experimental session in which the various configuration of the system are compared in three different domains. Moreover, since one of the main innovation with respect to the current literature is the reconfiguration via CSP, the experimental session dedicates a Section to assess how the reconfiguration technique scales when the number of the actions in the plan increases.

Chapter 7

Experimental Session

This Chapter studies the performance of FLEX-RR through an extensive experimental analysis on three different planning domains. The tests aim at evaluating the competence, the efficiency and the stability of the approach by putting particular attention on the reconfiguration, which is the main innovation w.r.t. the state of the art methodologies. Afterwards the Chapter will focus on the scalability of FLEX-R (i.e. the system without the Replan facility). The purpose is to understand the effectiveness and the limits of Recon when dealing with plans involving a large number of actions. Finally we conclude with a preliminary evaluation of the suitability of FLEX-R in handling optimization criteria.

7.1 Software and Hardware Setup

Figure 7.1 summarizes the environment implemented for the experiments. On the top (left) FLEX-RR takes in input the plan and the problem to be tested. On the bottom FLEX-RR interacts with an action simulator. This implements the execution of the action according to a modified version of its model (*MMAModel**), mimicking the evolution of the environment and in particular of the action execution. The modified version of the domain is obtained by noising the numeric effects of the action modalities.

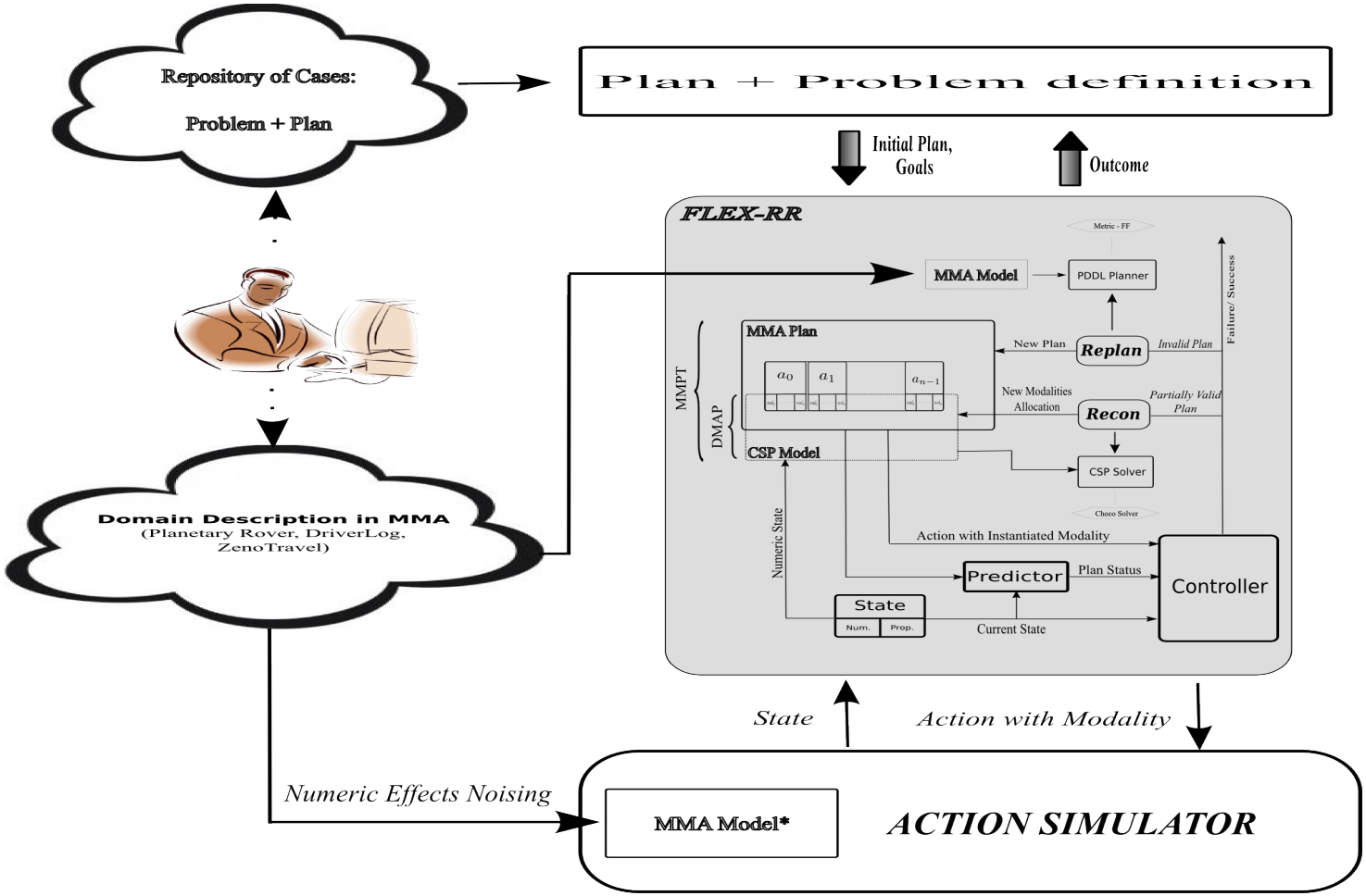


Figure 7.1: Experimenting FLEX-RR

FLEX-RR has been developed in Java 1.6, where the internal composition and the interactions is reported in Chapter 6. As input FLEX-RR receives the domain and the problem description written in an extended version of PDDL 2.1 level 2, which incorporates the notion of modalities. FLEX-RR exploits and extends the PPMAjaL library[1].

As a CSP solver we used Choco 2.1.4 [2]. the module is supposed to efficiently handle complex constraints on very large set of variables. As a planner we used Metric-FF ([50]) that supports the expressiveness of PDDL 2.1 level 2.

Experiments ran on a 2.53GHz Intel(R) Core(TM)2 Duo processor with 4 GB (OS: Ubuntu 10.04).

For the purpose of evaluating the system, we exploit three main FLEX-RR configurations:

1. FLEX-RR: the system with all the main submodules activated, i.e. the predictor module (for the detection of the plan validity), Recon and Replan.

2. FLEX-R: the system with the Replan switched-off. The DMAP is solved just by means of Recon

3. FLEX-REPLAN: the system with the Recon switched-off. The DMAP is solved via Replan.

7.2 FLEX-RR vs Replanning From Scratch

For evaluating the performance of the system in addressing the problem of the robust plan execution (see Chapter 3), we need to measure, on one hand, the effectiveness in handling unexpected deviations from the nominal behavior and, on the other hand the capability of the system in preserving, as much as possible, the plan structure despite the exception detected. To assess the Recon contribution, some test ran not only on FLEX-RR but also on FLEX-REPLAN and FLEX-R used as reference points.

In particular, our experiments have been performed by taking into account three main parameters:

- *Competence*: is the capability of a strategy of repairing a plan when the plan under execution becomes *partially valid*. This corresponds to a resolution of a DMAP. In particular, we measure the competence as the rate of successes in repairing a plan. In our experiments we allotted 200s to both FLEX-REPLAN and FLEX-RR as amount of time within which a solution must be provided; while FLEX-REPLAN can take the whole period

[1] See `www.di.unito.it/~scala`

[2] Choco is a java library for constraint satisfaction problems (CSP) and constraint programming (CP). Visit http://choco.emn.fr for any further information

(200s) for the replanning from scratch, we have subdivided the amount of CPU-time in FLEX-RR. More precisely, FLEX-RR gives just 30s to ReCon and 170s to the Replan. Thus, the competence of a strategy strongly depends on its efficiency in finding a solution. It is important to remark that FLEX-RR is supposed to work in an on-line context where typically the computational and temporal resources are limited. For this reason the actual competence of the system does not depend only on its theoretical capability but also on the efficiency in solving the task in a given amount of time.

Since we wanted to understand whether the reconfiguration performed by ReCon does contribute to the performance of FLEX-RR, we computed the competence also for the FLEX-R version (i.e. the system without the Replan facility). It must be noticed that, from a theoretical point of view, the FLEX-REPLAN competence is greater than the FLEX-R one; however, we guess that the greater efficiency of the reconfiguration compensates such a limit, since we will expect that while FLEX-REPLAN will be unable to find a repair plan given the time threshold, the reconfiguration will not.

- *Computational cost.* In particular we will compare the CPU time taken by the two strategies. This evaluation estimates with more accuracy the effective time employed to solve the impasse, i.e. the resolution of the DMAP. In principle, (as observed in Chapter 4) we will expect that the computational effort will be lower when the case can be solved via reconfiguration, since the reconfiguration task is theoretically simpler (in the worst case) than a replanning task (Chapter 4). This parameter will assess however what happens on the average.

- *Stability.* We will compare the plans produced by FLEX-RR with the ones generated by the only FLEX-REPLAN according to the stability metric defined in 4.5. We expect that FLEX-RR achieves, on the average, a better stability, since it focuses the search firstly on the space of modalities and just when this step fails it switches to replanning from scratch. Of course, this two steps process is aimed at privileging the structure of the old plan.

Tests were conducted in three challenging domains:

- *ZenoTravel*

- *DriverLog*

- *Planetary Rover*

The first two domains are from the Third International Planning Competition[3]. They have been introduced to challenge planners in handling numeric fluents. The third domain has been introduced in our recent works on intelligent supervision of space exploration missions [61, 53].

For each domain we generated a set of plans, each of which has been executed by means of the action simulator presented in Section 7.1. Numeric fluents modeling time and resources have been noised in order to reproduce unexpected contingencies. For each case, the injected noise is sufficient to alter the resource profile in such a way that each plan becomes *partially valid* at least once during the execution. In this set of experiments we have not taken into account situations where the plan became *invalid* as we were interested in evaluating the ability of FLEX-RR and FLEX-REPLAN in repairing *partially valid* plans.

7.2.1 Competence and Computational Cost

Experiments in the ZenoTravel domain

As introduced in 3.5 our version of the *ZenoTravel* domain extends the original definition by providing different execution modalities for each action[4]. In this domain we have collected 81 test cases, which vary not only for the different number of passengers and locations (from 4 to 13 for passengers and from 4 to 10 for locations), but also for the strictness of the constraints involving numeric fluents (i.e., *total-fuel-used*, the *cost* of boarding and disembarking, and the *total-time-spent*).

For all the 81 test cases we generated a plan whose length ranges from 18 to 44. Figure 7.2 shows the competence of the two strategies FLEX-RR and FLEX-REPLAN. As said above, to appreciate the contribution of ReCon, the figure also shows the successful rate of the FLEX-R configuration.

Note that, while FLEX-RR takes advantage of both ReCon and Replan and always finds a solution, FLEX-REPLAN has a lower success rate, this because in about 25% of the cases, FLEX-REPLAN was unable to find a solution within the threshold of 200 seconds of CPU time.

It is worth noting that the ability of FLEX-RR is mainly due to the performance of the ReCon facility. In fact, Recon alone was able to repair 99% of the cases.

In Figure 7.3 we compare the amount of CPU time used by FLEX-RR and FLEX-REPLAN. To provide a clear overview on the performance of the two strategies, we partitioned the allotted 200s of CPU time into 7 sub-intervals and the Figure shows the number of cases that have been repaired in each of these sub-intervals. It is easy to see that FLEX-RR solved most of the cases in

[3] http://planning.cis.strath.ac.uk/competition/

[4] For the complete domain definition see Appendix C.2

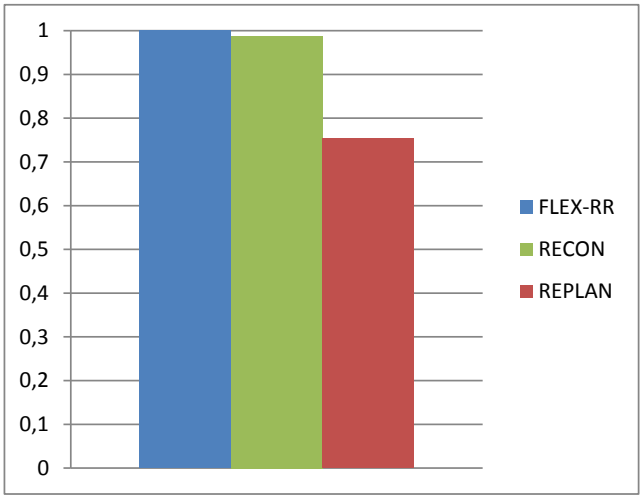


Figure 7.2: *ZenoTravel* Domain: Competence

less than 50 msec. On the other hand, the majority of cases solved by FLEX-REPLAN required less than 1 second of cpu-time; however, a significant number of cases (20%) was not solved within 200s.

Experiments in the DriverLog domain

The second domain used as test-bed is the *Hard-Numeric* variant of the *Driver-Log* domain. The reason that makes the problem hard is that, differently from the easy version, the fuel consumption does not depends only on the trip length, but also on how much the truck is loaded. We chose this version to stress ReCon in handling complex dependencies among numeric fluents.

To make the domain more interesting to our experimental setting, we enriched the domain by defining two modalities for the drive action (*fast* and *cruise*), which have an impact both on the fuel consumption and on the trip duration.

In addition, we added two different modalities for the *load-truck* (*safe* and *normal*), which differ each other in the amount of needed time[5].

In this domain we generated 60 cases varying the number of trucks and locations and the strictness of the constraints on the numeric fluents. The length of the resulting plans varies from 17 to 64 actions.

[5]The complete domain definition is reported in Appendix C.3

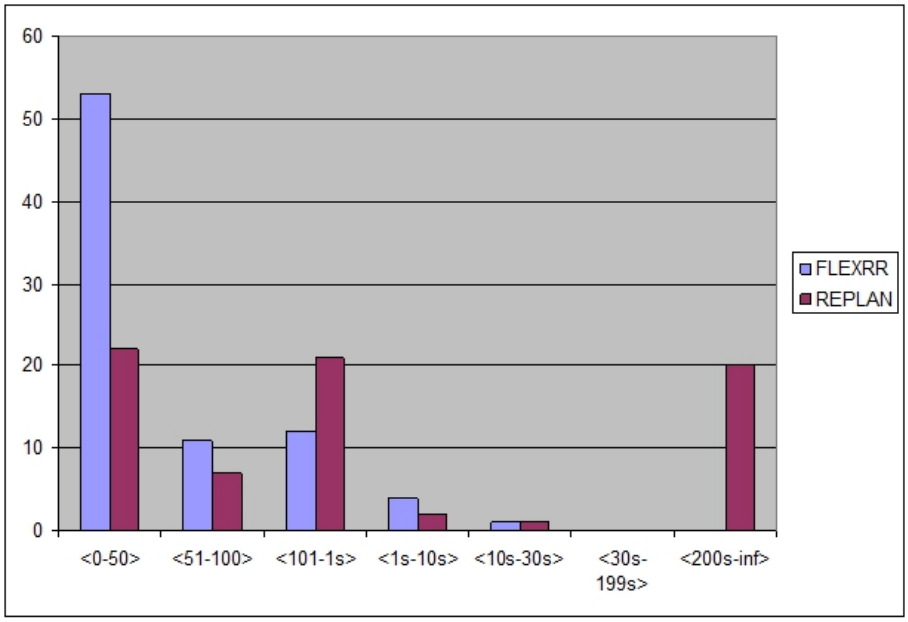


Figure 7.3: *ZenoTravel* Domain: Computational Cost

Figure 7.4 shows the competence of the two strategies FLEX-RR and FLEX-REPLAN. Also in this case, the Figure shows the competence of ReCon alone. It is easy to see that FLEX-REPLAN behaves quite inefficiently in this domain, while FLEX-RR performs quite well. In particular, observing the Figure 7.4, it is quite evident that a not negligible set of cases has been solved by FLEX-RR via REPLAN (the difference of competence between FLEX-RR and Recon). In these cases, we noticed that Recon has been always able to prove that the assigned DMAP had actually no solution, and thus the only way to overcome the impasse was via a Replanning mechanism. In few words, the Replanning has never been invoked for a timeout of the Recon.

FLEX-RR outperforms FLEX-REPLAN also from the computational point of view, see Figure 7.5. FLEX-RR solved most of the problems in less than 1s of CPU time, while FLEX-REPLAN solved just few of cases in such a time. Moreover, also in this case there is a significant number of cases that FLEX-REPLAN was not able to solve given the time threshold.

Experiments in the Planetary rover domain

The third domain we used as a test-bed is the *Planetary Rover* introduced in Chapter 3[6].

It is worth noting that this domain is very challenging for the CSP solver. On

[6]The complete domain definition is reported in Appendix C

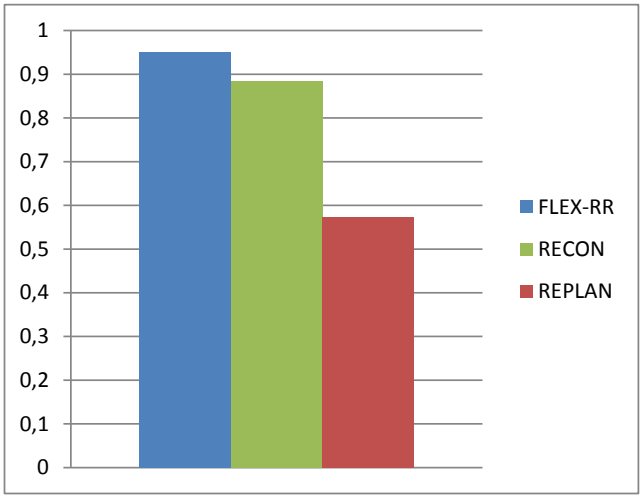


Figure 7.4: *DriverLog* Domain: Competence

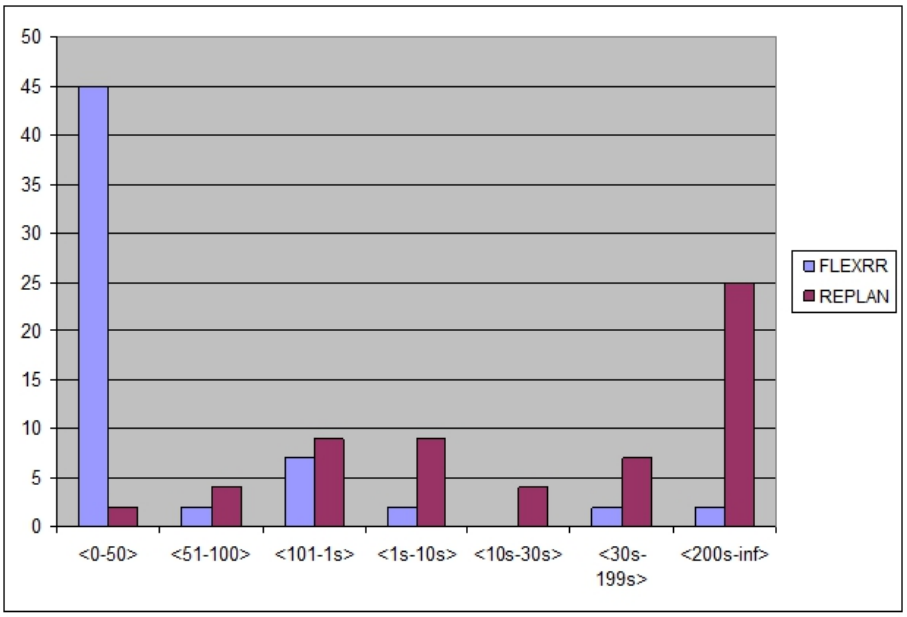


Figure 7.5: *DriverLog* Domain: Cpu Time

the one side, the domain takes into account many numeric fluents (e.g., the *total duration* of the mission, the *power* level available to the rover, the *total power consumption*, the *communication cost*, etc.). Thus there are many variables to

be taken into account. On the other hand, numeric fluents are strongly related to one another via the action models. For example, the *power spent* by the communication depends on the amount of information to be transmitted (i.e., *used memory*), which in turns depends on the resolution of the taken pictures.

In this domain we collected 81 cases with plans of 15-20 actions.

The 81 test cases were subdivided into two subsets: 36 *hard* cases and 45 *medium* cases. The classification depends on the degree of strictness we have imposed on the numeric constraints. In the *medium* cases the constraints have been imposed on time and power spent whereas for the *hard* cases the problem combines the previous constraints with others on the communication cost (which may limit the use of large bandwidth communication channel and therefore also to download a large amount of data in a short time) and the information loss (so that it is not possible to degrade too much the quality of the images and therefore the amount of data produced is high). Such further constraints makes the problem harder since they encodes opposite requirements, which strongly constrain solutions into limited portions of the plans space.

Figure 7.6 reports the comparison of success rate for FLEX-RR, FLEX-REPLAN and ReCon alone, for the medium and the hard cases, showing that FLEX-RR has high competence in both of them.

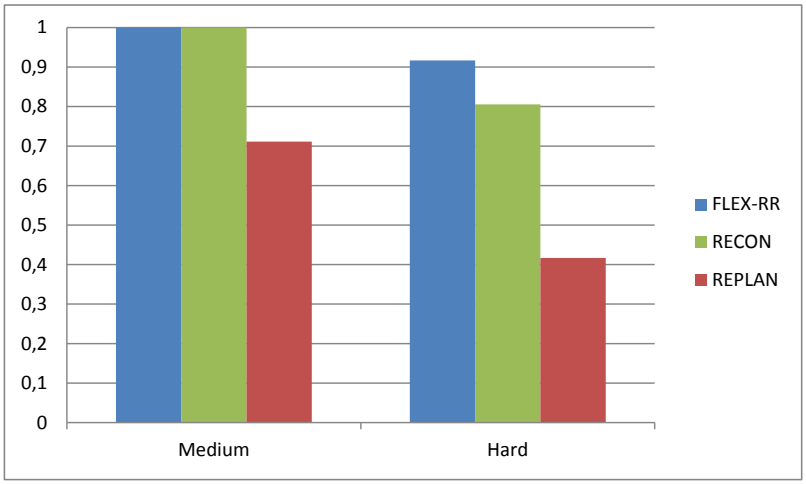


Figure 7.6: *Planetary Rover* Domain: Competence

The ability of ReCon alone to find a new plan decreases from medium to hard case (from 0,914 to 0,806), but in this case FLEX-RR can still obtain a

good competence by invoking the replanner. In particular, we measured that in the 50% of these cases (when the Replan is invoked), Recon has been able to provide the answer that the DMAP had no solution. In the rest of the cases, the Replanning has been invoked just after the detect of the timeout in the CSP resolution process.

The ability of FLEX-RR to deal with hard cases is further confirmed by analyzing the distribution of CPU time for solving the problem, see Figure 7.7. Despite the large number of modalities and numeric fluents defined in this domain, FLEX-RR is not only able to find a solution in less than one second for most of the medium cases, but also for many of the hard cases.

In conclusion, these results show that FLEX-RR, complementing a replanning with reconfiguration, is a viable solution to plan repair, and thus to enhance the robust execution of a plan.

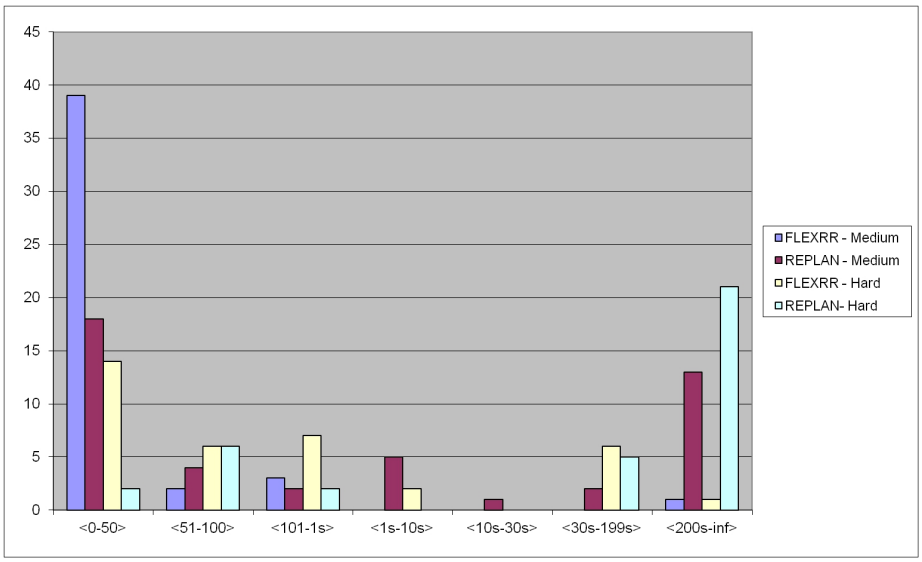


Figure 7.7: *Planetary Rover* Domain: Cpu-Time

7.2.2 Assessing the Stability

We evaluated the stability of plans repaired by FLEX-REPLAN and FLEX-RR in the same test set discussed in the previous Section. As reported in 4.5, the stability evaluation requires to set a cost for each operation (insert, remove, swap, and change of modality). We choose to weigh the insertion and deletion of actions (α) with a cost equals to 5, the change of the modality (γ) with 1, and the swap of two actions (θ) with 6.

Table 7.1 shows the results obtained in each domain, for the two configurations (FLEX-RR and FLEX-REPLAN). The third row, moreover, summarizes

the stability obtained for those cases where the REPLAN has been invoked in FLEX-RR because of a failure of Recon (for either timeout or not).

	Planetary Rover (Medium)	Planetary Rover (Hard)	DriverLog	ZenoTravel
FLEX-RR	0.907	0.807	0.971	0.97
REPLAN	0.709	0.646	0.791	0.50
FLEX-RR VIA REPLAN	##	0.40	0.85	0.62

Table 7.1: The Stability measured for each domain

In general, it is easy to see that FLEX-RR exhibits a higher stability grade than FLEX-REPLAN. For the domains *ZenoTravel* and *DriverLog*, the repaired plans produced by FLEX-RR (mostly by means of ReCon) exhibit a high stability grade, whereas the solutions provided by FLEX-REPLAN are quite different from the original plans. The hard cases in the *Planetary Rover* domain confirm to be very difficult. In fact, in this domain the stability grade decreases also for FLEX-RR. This happens because, in many cases, the solutions found by FLEX-RR are provided by the replanner (ReCon alone fails in finding a modality reassignment). In these situations, the resolution of the impasse has been possible only via a significant change of actions. By observing Table 7.1, the average stability obtained for the cases solved by FLEX-RR via REPLAN is in fact just 0.40. This proves in practice the validity of the Theorem 4, and hence our initial hypothesis. That is, FLEX-RR is able to obtain more stable plans than a replanner as FLEX-RR first tries to solve a DMAP, and only when the DMAP has no solution, FLEX-RR activates a replanner.

7.3 FLEX-R - Scaling Up Analysis

Having measured the performance of FLEX-RR against a mechanism based on a replanning from scratch step, this Section aims at evaluating whether and how ReCon scales when the number of MMAs increases, that is when the length of the plan to be reconfigured grows up. It is important to remember that in the worst case the search space for the CSP solver is exponential in the number of the actions to be reconfigured (see Section 4); in this Section we perform an analysis for the average case to understand when the Recon mechanism is still effective.

The configuration tested is FLEX-R (we switch off the replanning facility) and the benchmark is performed on the *Planetary Rover* and the *DriverLog* domain.

Differently from the previous setting, to observe the behavior of the system

for several and increasing (in the number of the actions) instance of DMAP, we start from a plan computed directly by Metric-FF through a series of random instances of planning problems. The only information seeding the generation of the problem has been the number of sites (for *Planetary Rover*) and of cities (for *DriverLog*). The rationale is to try to control the size of the resulting plan. Intuitively the greater the number of locations/sites is, the longest (on the average) the plan would be[7].

The number of cases collected amount to 405 for the *Planetary Rover* domain (where the plans range from 1 to 50 actions) and 225 for the *DriverLog* domain (1 to 75 actions). For both benchmarks we evaluate the cpu-time needed (with a 60 secs limit) to the solver to provide an answer. Differently from the evaluation performed in the previous Section, we consider a success not only the fact that the system has been able to solve the impasse (the competence), but also the case where the system was able to provide a negative answer (there is no solution for the given DMAP). The failure for the system is hence when the timeout is reached.

Similarly to the previous Section, to obtain deviations on the use of resources, we adopted a modified action model containing noised numeric effects.

Results have been organized categorizing the tests w.r.t. the length of the DMAP - the only actions to take care of are those still to be executed - and the percentage of cases belonging to a given time interval. More precisely, as far as it is concerned by the length of the DMAP, we distinguish 5 classes of problems (0-10,11-20,..,41-50) for the Planetary Rover and 5 classes of problems for the *DriverLog* case (0-15,...,61-75). For each class of problems we discern 6 different *performance* intervals:

1. Very Fast :: 0-100 ms
2. Fast :: 101-1000 ms
3. Normal :: 1001-5000 ms
4. Slow :: 5000-15000 ms
5. Very Slow :: 15000-59999 ms
6. Failure :: greater than 60000 ms

Planetary Rover Domain

Observing the histogram of the *Planetary Rover* domain reported in Figure 7.8 it is clear that the FLEX-R behaves quite well as the most of the cases are

[7]The length of the plan depends on many other factors which actually rely on the nature of the problem and the domain of application (e.g. kinds of actions at disposal, number of goals and so forth); in Appendix B and B.2 we show how the generation is performed for these two domains.

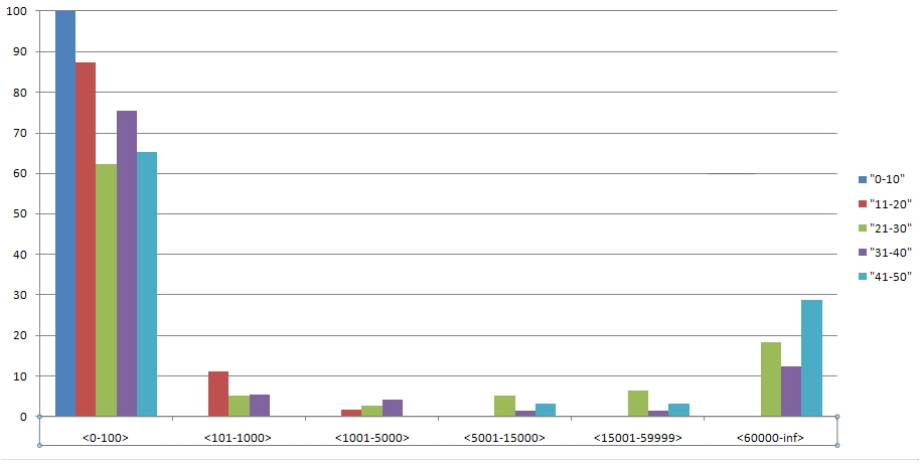


Figure 7.8: Scaling Up Tests for the Planetary Rover Domain. The y-axis refers to the percentage of successful cases for each interval of interest (abscissa)

computed in the first time interval ($0 < 100$). In particular this holds for the resolution applied for plans whose length is less or equal to 10 actions. For this class of cases the mechanism provided an answer in less than 100 ms for all the submitted cases.

The situation remains quite stable until 20 actions, getting worse only when the plan contains more than 40 actions. Indeed, reported on the right part of the histogram, the percentage of failing situation for the class "41-50" actions is estimated almost in the 30 % of situations.

In general the results proved that until 20 actions FLEX-R is very efficient. As a matter of fact, only few cases exceeds 1 seconds (the red stack on the third interval). Whereas, on the average, the categories with more than 20 actions fails some time (10% to 30 %). However, also for this classes, there are a non negligible set of cases that were solved in less than 100 ms. (60-75%).

As a demonstration of the complexity of the domain, the CSP structure generated by the Choco solver for this set of cases contains on the average 1500 variables and 1783 constraints.

DriverLog Domain

In the *DriverLog* domain, the results continues to be rather positive. In this domain, the system has been stressed until plans involving 75 actions. The motivation behind this is that, differently from the *Planetary Rover*, the *Drivelog* domain includes actions with just one modality. Therefore, on an equal number of actions, the search space of the DMAP for the *DriverLog* domain turns out to be smaller than the one of the *Planetary Rover* domain.

By comparing Figure 7.9 and 7.8, it is quite evident that, as the length of

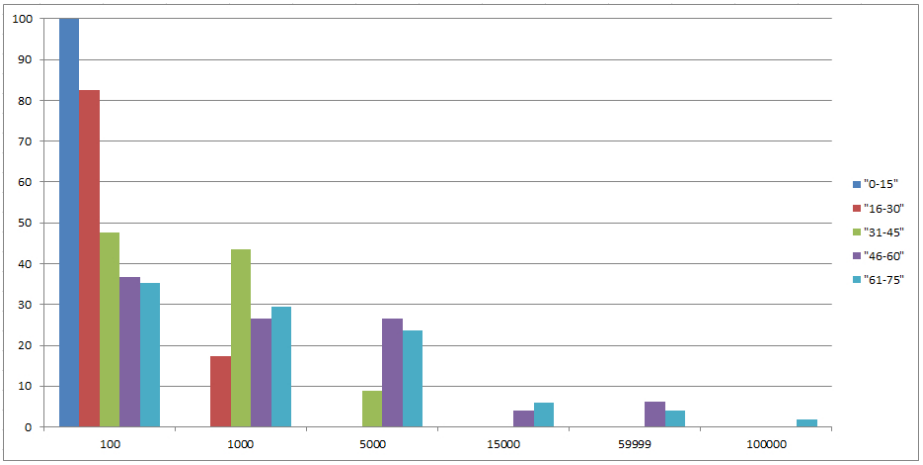


Figure 7.9: Scaling Up Tests for the *DriverLog* Domain

the plan increases, the DMAP behavior for the *DriverLog* seems smoother than the behavior of the DMAP for the Planetary Rover task.

Similarly to the Planetary Rover the first class of difficulty (from 0 to 15 actions) behaves with extreme efficiency (for this class of plans we solve the DMAP in less than 100ms in the 100% of cases), while in the *DriverLog* instead the performance of the system degrades more slowly. In particular only two plans with actions included between 61 and 75 have not been solved within the time threshold.

Also in this case, the structure of the CSP representation is significant. We measured, in fact, on the average, 838 variables and 923 constraints for the DMAPs considered.

7.4 Beyond FLEX-RR :: Handling Optimization Criteria

In the previous Sections we have substantially analyzed the capability of ReCon in providing a solution for the original version and purposes of the DMAP. The objective is to provide a decision as soon as possible, for this reason the DMAP only cares about the feasibility of such a solution. That is, FLEX-RR via the CSP Solver picks just the first valid solution that encounters during the search. However, in principle, a given MMA plan can contain many possible valid modality allocations. For this reason this Section pursues a slight different objective and evaluates the ReCon task when a certain criterion of optimization is specified over the resources involved.

We start from the observation that, while the original DMAP is well suited in an on-line context, in an off-line situation and hence before the plan execution,

the user can be interested in evaluating a given plan of MMAs for understanding the limit under which such a plan is requested to work (e.g. for example the minimum time that could be spent). That is, it could be useful to analyze the plan w.r.t. the maximum (or the minimum) cost specified on some particular resource.

To perform the experiments, we oblige the CSP-solver to prove not only that the solution is consistent w.r.t. the constraints defined in the problem, but also that such a solution minimize (or maximize) a given numeric fluent (or a combination of them). Such a criterion is expressed on a specific numeric fluent of our problem, and in particular the solver optimizes the value that it takes at the last snapshot of the CSP.

Tests ran on the same benchmark suite built for the scaling-up (previous Section). For the *Planetary Rover* domain, we constrain the solver to find the solution which minimizes the overall cost of the communication, for the *Driver-Log* domain, we configured the solver to search for the solution that minimizes the total fuel used. Moreover, we do not allow to express both the optimization and the hard constraint of the DMAP on the same numeric fluent.

The organization of the data is equivalent to the schema used for the previous Section. As a difference, here the system is evaluated w.r.t. the capability of proving that the found solution (if any) is optimal. Therefore, the failure situation does not necessary mean that the system did not find any solution. Indeed, Choco employs an anytime search strategy; thus, in many cases, it was able to provide a valid solution (the best one at the end of the computation), even if this was not optimal.

As for the scaling up, we imposed a cpu-time limit of 60 seconds.

Planetary Rover Domain

Figure 7.10 points out the benchmark results for the *Planetary Rover* domain.

The experiments are quite interesting. As reported by the histogram, the system behaves in a similar way to the scaling-up benchmarks. Of course nothing comes for nothing, and the class of problems solved efficiently shrinks to the class of plans whose is "0-10" actions. Differently, already for the class of "11-20" actions the mechanism requires some extra time. This can be observed by looking to the number of cases in the second and third intervals of time.

However the percentage of success is still quite high in all the cases tested.

DriverLog Domain

Figure 7.11 shows the results for the *DriverLog* Domain.

The benchmark presented for the *DriverLog* proves that until 15 actions the mechanism is still very efficient. As a matter of fact, for the first class of

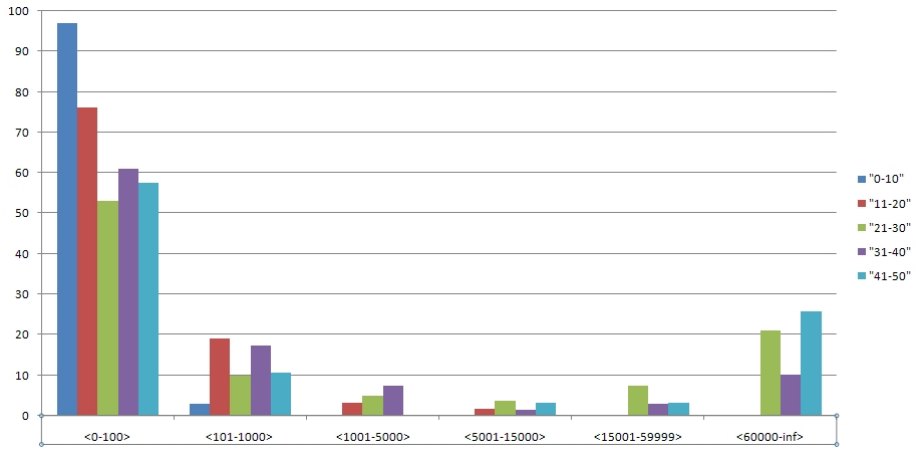


Figure 7.10: Scaling Up Tests for the Planetary Rover Domain. Optimizing communication cost

difficulty the solver was able to find the optimal solution in less than 100 ms in all the cases.

The performance of the approach degrades only when the number of the actions to consider is greater than 60. However, till the 45 actions just the 2% of cases required 15 seconds of cpu-time while the most of cases have been solved with a maximum of 5 seconds.

7.5 Conclusions

This Section reported an experimental session for measuring the effectiveness and the performance of the FLEX-RR architecture.

The experiments have been performed on three domains: *Planetary Rover*, *ZenoTravel* and *DriverLog* domain.

In a first phase we measured the competence, the performance and the stability of three configurations of FLEX-RR: (i) the system with all the facility activated, (ii) FLEX-R which is the architecture with the only Reconfiguration functionality activated, and (iii) FLEX-REPLAN which is the configuration provided with the Replanning but not with the Reconfiguration feature. This last configuration corresponds to a traditional system without the main contribution of this thesis.

Results showed that the FLEX-RR is more efficient than FLEX-REPLAN (FLEX-RR outperformed FLEX-REPLAN in all benchmarks) and, since the latter often could not provide a solution given the time threshold, FLEX-RR turns out to be more competent too. This confirms the reconfiguration contribution to the FLEX-RR architecture, and our hypothesis that re-configuring MMA is more convenient that replanning from scratch. As refers the *stabil-*

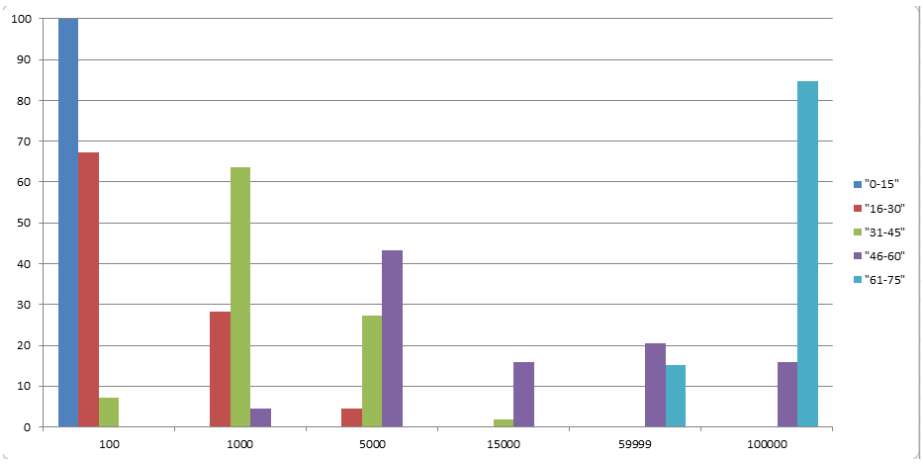


Figure 7.11: Scaling Up Tests for the DriverLog Domain. Optimizing fuel consumption

ity, we have the confirmation that FLEX-RR keeps the plan more stable than FLEX-REPLAN and, as anticipated by Theorem 4, the *stability* degrades only when there is no alternative allocation of modalities (i.e., when the DMAP has no solution).

In a second phase we studied the FLEX-R scalability. The results have showed that in the *Planetary Rover* domain FLEX-R is very efficient till plan of 20 actions continuing to perform quite good also in longer plan (31-40 actions). Similarly, in the *DriverLog* domain FLEX-R turns out to be quite efficient until 30 actions. The performance degrades only for plan with a number of actions greater than 40.

Finally we measure FLEX-R in dealing with optimization criteria both on the *Planetary Rover* and the *DriverLog* domain. Even beyond the original aims of the FLEX-R, the results prove that the system can be useful in an off-line phase, too; for instance as a decisional support tool for a user, who could be interested in evaluating a given MMA plan in terms of maximum (minimum) resources usage.

Chapter 8

The Role of Numeric Kernel for Plan Involving Numeric Fluents

This Chapter introduces the notion of *numeric kernel* as a means for reasoning about plans involving numeric state variables, i.e. numeric fluents. The *numeric kernel* notion can be employed to improve the FLEX-RR mechanism. Moreover, applied to the context of MMAs provides a way for integrating the plan supervision task offered by FLEX-RR with a more reactive control as ACTS (see Chapter 9). We believe the notion is quite general in the context of the automated planning and an application of this concept has been already presented in our recent work [90].

8.1 Introduction

As we have seen in Chapter 6, the on-line execution of a plan requires the agent to be progressively aware whether the plan is still feasible given the state conditions encountered/observed. Unfortunately, the applicability conditions embodied in the preconditions set of the next action to be executed are in general not sufficient to infer whether the current course of actions still leads towards the goal. For this reason, and in particular in the FLEX-RR, we adopted a standard technique based on a simulation step (the recursive plan execution starting from the current state) to understand whether the final predicted state is reacheable with the plan at hand.

In the context of propositional planning (in particular in the STRIPS fragment [40], [43]), it has been however noticed that, by observing a given plan of actions and the set of goals it is possible to precompute a set of sufficient

and necessary conditions which allow to immediately assess the plan feasibility. Such conditions has been called kernel. The propositional kernel has been indeed employed in the so called PLANEX system where a backward method, based on Triangle Table has been defined.

In this Chapter, we generalize the notion of the kernel to the numeric setting; that is, when the plan is requested to deal with numeric information as resources, counters, costs and so forth. More precisely we introduce the notion of *numeric kernel*, which is a set of comparisons involving numeric fluents from the domain. Analogously to the propositional case, the *numeric kernel* formulation aims at establishing the sufficient and necessary requirements that must be guaranteed for the executability of a plan obeying to particular numeric profiles (e.g., resource constraints). More precisely, a *numeric kernel* is supposed to capture the characteristics of a state of the system such that, whether the plan is executed from such a state, then the (numeric) goals is achieved. While a STRIPS kernel expresses exactly the minimal set of propositional atoms that must hold during the plan execution, here, the *numeric kernel* does not specify any particular state configuration. Instead, the *numeric kernel* expresses just the intervals of numeric variables that are mandatory for the consistency of the plan.

As we will see the *numeric kernel* can be computed by extrapolating knowledge from the problem, the domain and the plan under consideration. The idea of keeping trace of the action model combined with the specific planning problem at hand (in particular with the goal), is not completely new, the propositional kernel developed by [40] has a similar construction.

Even if the term has been in principle coined just for the STRIPS setting, we will refer to the kernel as the combination of the propositional and the numeric one [1]. Indeed, a numeric planning task (as the one defined in our system, see Definition 8) mixes propositional and numeric aspects. For this reason, in order to evaluate the feasibility of a plan as a whole, both characteristics have to be considered.

The utility of the only propositional kernels has been proved to be a viable solution not only for the monitoring problem (in particular when the agent has to assess the validity of the plan being executed), but also for repair strategies (when some unexpected condition arises). Recently, the system developed by Garrido et al. ([44]) has proposed a (propositional) kernel based repair technique.

Relying on the notion of numeric kernel, the Chapter will present three possible enhancements for FLEX-RR:

- **Improved Monitoring**: a novel technique to infer in an efficient way

[1] In the rest of the discussion we will specify whether we will focus on the propositional or the numeric counterpart.

whether the plan is either *valid* or *partially valid* or *invalid*;

- **Safe Control Delegation**: a method to individuate a set of *safe execution modalities* in the applicability conditions for an MMA[2]
- **Kernel Replanning**: a focused replanning mechanism to be applied for *invalid* plans[3]

For the sake of the explanation, the Chapter mainly focuses on actions having just one modality of execution. Indeed, both the monitoring and the replanning via kernel do not require that actions are expressed in form of MMAs[4]. The notion of safe execution applies only for multi modality plans, therefore, when we introduce the concept, the kernel will be combined to the notion of the MMA.

The Chapter is organized as follows; firstly we formally define the notion of the *numeric kernel* (Section 8.2), whereas in Section 8.3 we explain and show how *numeric kernels* can be computed. Section 8.4 describes how and when the *numeric kernel* can be employed to improve FLEX-RR. The Chapter ends with an experimental Section which shows the adoption of the kernel in a suite of test cases for the *ZenoTravel* domain (Section 8.5).

8.2 Numeric Kernel

Similarly to the propositional case, the main role of the *numeric kernel* is to specify particular requirements to be used for inferring if a state *supports* the achievement of the goal, through the execution of a given plan. In other words, a kernel individuates the sufficient and necessary conditions for a plan such that, given a state S obeying to such requirements, the plan can be successfully performed starting from S.

Before of formally defining the Numeric Kernel, let us introduce the propositional kernel definition ([40]) adapted for our purpose. That is:

Definition 15 (Propositional Kernel)**.** *Given a plan π solving a Multi-Modality Planning Problem (MMPP) $\Pi = < A, I, G_{prop}, G_{num} >$ for the domain $< U, F, X >$, and K a set of propositional atoms such that $K \subseteq F$, K is said to be a propositional kernel for π and G_{prop} if, given a state S, $S[\pi]$ satisfies G_{prop} iff S_{prop} satisfies K.*

[2]As we will see this step is fundamental for combining the plan supervision task with a more reactive control, i.e. Action Supervision. See Chapter 9.

[3]Let us remember that *Recon* can be applied just in case the plan is partially valid. When the plan is not valid anymore or even there is no solution by Recon, a replanning mechanism is mandatory

[4]It is quite interesting to see that an MMA with just one modality of execution can be seen as a common PDDL 2.1 level 2 action ([42]). For this reason the *numeric kernel* can be employed without modification for the numeric PDDL setting, too.

Concretely, a propositional kernel is a set of atoms which corresponds to the minimal conditions necessary for the validity of a plan.

The *numeric kernel* generalizes the propositional kernel formulation to support numeric information, expressed in form of numeric fluents. In our context, the numeric fluents are elements from X, which are allowed to appear as terms in the preconditions and effects of the actions, as well as in the problem statement (see Definition 8). They constitute the basic elements of the arithmetical expressions involved in the comparisons (used in the action precondition and the goal statement) and in the assigners (numeric action effects).

To handle such a hybrid nature of the kernels we are going to propose, we differentiate the numeric and the propositional part by subscripting the K symbol. Specifically, K_{prop} will identify the propositional kernel while K_{num} the numeric one.

Thus, a *numeric kernel* K_{num} can be formally defined as follows:

Definition 16 (Numeric Kernel)**.** *Given a plan π solving a Multi-Modality Planning Problem (MMPP) $\Pi = < A, I, G_{prop}, G_{num} >$ for the domain $< U, F, X >$, and K_{num} a set of inequalities of the form $\{exp, \{<, <=, =, >=, > \}, exp'\}$ built over X, K_{num} is said to be a* numeric kernel *of π iff it represents a set of sufficient and necessary conditions for the achievement of the numeric part of the goal G, i.e. G_{num}, starting from S. That is, given a state S, $S[\pi]$ satisfies G_{num} iff S_{num} satisfies K_{num}.*

where a state S satisfies a set of inequalities K_{num} if the assignment of the numeric fluents in S is consistent with each inequalities in K_{num}. In other words, the assignment of the numeric fluents in S is one of the solutions for the system of inequalities expressed by K_{num}.

These are comparisons among group of expressions involving numeric fluent from X. Thus they have the same form of the numeric precondition expressed in Definition 3. In the next we will prefer the use of the term comparison instead of inequalities.

By considering the definition of state reported in 1 (Chapter 3), the numeric part of the state can be represented as a vector in a multi-dimensional space where each numeric fluent represents a specific dimension. An hypothetical *numeric kernel* as defined above represents a region (potentially unbounded in some dimensions) of the states space which is consistent with the *numeric kernel* itself.

By considering each suffix of the plan $\pi = \{a_0, ..., a_{n-1}\}$, i.e. $\pi_1 = \{a_1, ..., a_{n-1}\}...\pi_2 = \{a_2, ..., a_{n-1}\}...\pi_{n-1}$ $\{a_{n-1}\}$ till the empty plan $\pi_n = \{\}$, it is possible to individuate an ordered set of *numeric kernels* in which the i-th element set is the *numeric kernel* of π_i. It is worth noting that, by definition, the goal is a special kind of *numeric kernel* for the empty sub-plan.

Finally, given a plan of size n it is possible to identify $n + 1$ kernels where:

- $S^0[\pi_0]$ satisfies G_{num} iff S^0_n satisfies K^0_{num}
- $S^1[\pi_1]$ satisfies G_{num} iff S^1_n satisfies K^1_{num}
- ...
- $K^n_{num} = G_{num}$ corresponding to the kernel for an empty plan

In the definition above the superscript indicates the "time" index of interest.

As we will see in the next Section, the creation of the *numeric kernels* can be done just once in a pre-processing phase before the real plan execution. Once obtained the kernels set, one of the interesting property is that the verification process becomes very easily; indeed it can be performed by simply substituting the numeric values of the state in each comparison appearing in the kernel. For this reason one can immediately infer whether the goal is still supported by the current state. Of course in case the plan undergoes some adjustments the set of kernels has to be recomputed.

Given a kernel K according to Definitions 16, 15 and the plan consistency notion reported in Definition 10, it is possible to deduce that:

Proposition 5. *A plan π is valid at step i for a goal G iff S^i satisfies K^i, where*

- *S^i is the state observed before the execution of the subplan π_i*
- *K^i is the i-th kernel of π*

Proof. $\Longrightarrow$
Given S^i and G, If the plan π is valid at step i we are sure that $S^i[\pi_i]$ satisfies both G_{prop} and G_{num}. However, by definition of kernels, the only states supporting the plan to achieve the goal are the ones satisfying K^i; for this reason it follows that both S^i_{prop} satisfies K^i_{prop} and S^i_{num} satisfies K^i_{num}. Meaning that S^i satisfies K^i.

$\Longleftarrow$
By definition of propositional and *numeric kernel* we know that if S^i satisfies K^i we are sure that: (i) S^i_{prop} will contain the necessary atoms - indeed S^i_{prop} satisfies K^i_{prop} - for the achievement of the goal G_{prop} via π_i, hence $S^i[\pi_i]$ will satisfies G_{prop}; (ii) S^i_{num} satisfies the inequalities expressed in K^i_{num}, hence $S^i[\pi_i]$ will satisfies G_{num}. Hence it follows that π is valid at step i as $S^i[\pi_i]$ satisfies both G_{prop} and G_{num}

□

The proposition reported above, highlights the very strong interdependence among the kernel and the plan consistency status. This proposition is indeed exploited by FLEX-RR for evaluating the validity conditions of the plan. See (Section 8.4).

Before explaining the construction of the kernel, let us introduce a small example which focuses on the numeric part of a possible planning task.

8.2.1 Example

Suppose a very simple version of our *Planetary Rover* domain in which there is just one action "drive" that decreases the power at disposal, and let us imagine to have the following situation:
$X = \{(power)\}$
$\pi = \{(drive)\}$
$G = \{((power) > 7)\}$

$$drive = \begin{cases} pre_{prop} = null \\ pre_{num} = null \\ eff_{prop} = null \\ eff_{num} = \{((power) - = 5)\} \end{cases}$$

Observing the setting it is easy to see that the sufficient and necessary conditions for making applicable with success the plan π should constrain in some way the value of the fluent $(power)$. That is, the state which supports the successful execution of this plan has to satisfy some particular condition on the value of the power. In particular, the kernel [5] imposes the $(power)$ to be greater than 12. So $K^0 = (power) > 12)$. By applying the action indeed, it is easy to see that it is possible to achieve the goal only in case $(power)$ at the initial state is greater than 12 . Otherwise, a value less or equal to 12 would have compromised the feasibility of the plan, since the final state of $(power)$ would have been a number less or equal to 7. In this case the action preconditions are empty, so the only necessary conditions to keep trace of, are the ones belonging to the goal conditions. If it had not been the case, the kernel would have considered also the applicability condition of the action. For instance, if the action requires that $(power\ \ r1)$ is at least 7, the kernel has to contain also additional constraints. That is $K^0 = \{((power) > 7), ((power) > 12)\}$, which of course can be simplified yielding $K^0 = \{((power) > 12)\}$.

8.3 Kernel's Construction

This Section explains the construction of the set of *numeric kernels*. Analogously to the previous Section, for completeness, we firstly show the computation of the propositional kernel. It is worth noting that the two processes can be performed independently as it is sufficient to combine the two results at the end of the computation.

[5] In this case the propositional kernel would be empty as there are no propositional information

Even if the kernel's definition focuses on just one specific kernel, that is the one useful for assessing the validity of the plan, the construction we are going to propose is intended to generate the entire ordered set of kernels useful for each suffix of the plan. In a nutshell, the purpose is to build a structure that allows the agent to be aware of the plan consistency for each step of the execution.

8.3.1 Propositional Kernel Construction

Analogously to the procedure reported in [40] via the Triangle Table, the propositional kernels are built by a backward propagation that keeps trace of the action model information. The algorithm 6 explains the process in detail.

Algorithm 6: Propositional Kernels Computation (PKC)

Input: $\pi = \{a_0,..,a_{n-1}\}$ - plan;
G - goal
Output: K: an ordered set of propositional kernels
1 $K^{|\pi|}_{prop} = G_{prop}$
2 **for** i=$|\pi|$-*1* to *0* **do**
3 $\quad K^{i}_{prop} = \{K^{i+1}_{prop} \setminus eff^{+}(a_i)\} \cup pre(a_i)$

The procedure starts from the last (trivial) kernel corresponding to the set of the goals (the atoms that must be achieved at the final state, line 1, and produces each i-th kernel by (i) removing the atoms provided by a_i (i.e. the add-list of the i-th action), (ii) adding the atoms required by a_i (i.e. the propositional preconditions of the i-th action), line 3.

8.3.2 Numeric Kernel Construction

As we have seen in the Definition 16, differently from the propositional case, the *numeric kernels* depend on the numeric part of action models (more precisely the ones that are employed in the plan) and on the numeric part of the goal conditions. As anticipated indeed, each *numeric kernel* K^{i}_{num} will define the conditions for a state S to be a valid state for the achievement of the numeric goal by applying a piece of the plan π, i.e., π_i.

Analogously to the propositional case, the main idea behind the construction process is to keep trace of the action effects (in particular the numeric ones), while preserving its preconditions all along the plan, starting from the goal conditions. The main steps are reported in the algorithm 7

The high level formulation of the algorithm is quite similar to the propositional one. In particular, the algorithm starts with the last *numeric kernel*, i.e. the goal conditions. Then, the process constructs the (previous) *numeric kernel* iteratively by combining the information involved in the numeric part

Algorithm 7: Numeric Kernel Computation (NKC)

Input: $\pi = \{a_0,..,a_{n-1}\}$ - plan;
G - goal
Output: K: an ordered set of numeric Kernels
1 $K_{num}^{|\pi|} = G$
2 **for** i=$|\pi|$-*1* to *0* **do**
3 $\quad K_{num}^{i} = \{K_{num}^{i+1} \oplus eff(a_i, m)\} \cup pre(a_i)$

of the action model and in the next *numeric kernel* (line 3), which has been previously computed. That is, the computation of K_i depends on a_i and K_{i+1}.

The main difference w.r.t. the computation of the propositional case, and hence the main contribution of this Chapter, refers to the $\oplus$ operator.

The operator $\oplus$ is a function that maps a set of comparisons (of the form $exp, \{<, \leq, =, \geq, >\}, exp'$) and a set of assignments to a new set of comparisons[6]. More formally, the operation $\oplus$ performs the steps reported in algorithm 8.

Algorithm 8: $\oplus$

Input: Eff: numeric effects of a
C: a set of comparisons
Output: C': a set of comparisons
1 $C' = \{\}$
2 **foreach** c $\in C$ **do**
3 $\quad c'_l = c_l$.sbt($x_0...x_{n-1}$,$eff_{x_0}...eff_{x_{n-1}}$)
4 $\quad c'_r = c_r$.sbt($x_0...x_{n-1}$,$eff_{x_0}...eff_{x_{n-1}}$)
5 $\quad C' = C' \cup \{c'\}$

The algorithm takes in input the set of comparisons and the numeric effects of the action a. c_l identifies the left part of the comparison while c_r the right one. Thus both c_l and c_r are arithmetical expressions over the numeric fluents of the problem.

For each comparison in C, the algorithm performs a substitution of the numeric fluents involved in c_l and c_r, according to the assignments reported in a. For instance, if the numeric effects of a affect a fluent x with $increase(x, 5)$ and the comparison asserts that $x < 4$ the outcome C' will be $x + 5 < 4$, i.e. $x < -1$. Of course, the action can affect all the fluents involved in the previous comparison; therefore the substitution must map each fluent involved in C with the effect described in the action model.

Each eff_{x_i} is the numeric effect of an action, where x_i is the numeric fluent that changes by means of the action application. Let us remember (Definition 3) that a numeric effect is defined by means of the triple $< op, f, exp >$ so x_i is

[6]Note that every operation in the action model can be transformed in an assignment operation.

the f and represents the "future" while exp is an arithmetical expression over the fluents representing the "past". This expression can be evaluated in the state in which the action is applied.

By observing Definition 3 it is easy to see that comparisons and effects can encompass potentially complex arithmetical operations among the numeric fluents. Similarly to what has been defined in the action model, we restrict the expression to be linear expressions (i.e a linear combination prevents the occurrence of operations such as f_1 $\{*,/\}$ f_2).

For simplicity, let us introduce a small example. Let us imagine to have an MMA, i.e a, where there is just a single modality of execution m, defined numerically as follows:

$$\text{pre(a,m)} = \begin{cases} f_1 > 5 \\ f_2 < 4 \end{cases} \quad \text{eff(a,m)} = \begin{cases} f_1 = f_1 + 5 \\ f_2 = f_2 + 8 \end{cases}$$

and a set of comparisons C as follows:

$$\text{C} = \begin{cases} f_1 > 10 \\ f_2 < 4 \end{cases}$$

The operation $\oplus$ between a and the comparisons involved in C (i.e. $a \oplus C$) will transform the assignments defined in eff(a,m) and the comparisons in C producing a new set of comparisons. More precisely, in the example above we will have:

$$C \oplus eff(a,m) = \begin{cases} f_1 + 5 > 10 \\ f_2 + 8 < 4 \end{cases}$$

By continuing on this example, we can see a specific iteration of the algorithm 7 for the action a.

Let us imagine that the plan π consists of just an action, i.e. $\{a\}$, and that the goal is represented by the same inequalities of C.

The *numeric kernel* for the plan π given C is obtained by joining the constraints defined by $\oplus$ with the constraints defined for *pre(a,m)* hence:

$$(C \oplus eff(a,m)) \cup pre(a,m) = \begin{cases} f_1 + 5 > 10 \\ f_2 + 8 < 4 \\ f_1 > 5 \\ f_2 < 4 \end{cases}$$

which can be simplified yielding:

$$(C \oplus eff(a,m)) \cup pre(a,m) = \begin{cases} f_1 > 5 \\ f_2 < -4 \end{cases}$$

It is easy to see that if we take an arbitrary state with f_1 and f_2 under the comparisons defined above (e.g. $f_1 = 6$ and f_2 = -6), and if we apply to such a state the plan π we are sure that (i) the action is applicable, since $6 > 5$ and $-6 < -4$, and (ii) we will obtain a state S in which both $f_1 > 10$ and $f_2 < 4$ hold, namely the goal will be satisfied.

The example reported above describes a simple scenario where numeric fluents do not depend on each other. But this is not always the case. Theoretically, in fact, an effect for the action can express that $f_1 = f_2$. Also this kind of representation is captured by the substitution performed by the algorithm 8. In general the algorithm will substitute each variable of the comparison with the way in which the variable is modified (line 3 and 4 of algorithm 8).

Let us conclude this Section with the proof of the correctness of the algorithm 7. For the sake of explanation, the correctness proof focuses on the numeric aspect of the problem, so G, S, K and the plan π are analyzed as far as it is concerned by just their numeric part.

Theorem 6. *Correctness. Given a plan π and a goal G the algorithm 7 finds a set of* numeric kernels *for π and G.*

Proof. The proof proceeds by induction on the length of the plan.

The base case of our induction is when the plan is empty, i.e. $|\pi| = 0$; in such a case the algorithm will compute just one set of comparisons, namely K^0, which is the one containing the same comparisons present in the goal. For this reason, given a state S, it follows that $S[\pi]$ satisfies G if and only if S satisfies K^0. In particular $S[\pi]$ corresponds to S and the only way of satisfying the goal is to be a state that already satisfies the goal conditions.

Inductive step. For inductive assumption we know that the i-1 steps of the algorithm KC (the iteration) have computed a set of i *numeric kernels* for a plan of length n; i.e. we have the set of kernels K = $\{K^{n-(i-1)}, K^{n-(i-2)}..., K^n$ $\}$ which is in relation with each suffix of the plan, i.e. with $\pi_{n-(i-1)}$,$\pi_{n-(i-2)}$...,π_{n-1}[7]. At the i-th step, K^{n-i} is computed by combining the precondition of the first action in π_{n-i}, i.e. $\{a_i\}$ and the comparisons obtained in the previous step. For this reason, a state S satisfying K^{n-i} will be such that (i) the action $\{a_i\}$ is applicable and (ii) the state resulting from the action application, i.e. $S[a_i]$ turns out to satisfy the $K^{n-(i-1)}$. This last in fact follows directly from the definition of $\oplus$. Indeed the operation keeps the conditions expressed in $K^{n-(i-1)}$, while considering the effects of a_i by means of the substitution mechanism. Having both (i) and (ii) we are sure that if S satisfies K^{n-i} , then $S[\pi_{n-i}]$ will satisfy the goal.

For the *necessary conditions*, that is if $S[\pi_{n-i}]$ satisfies G then S satisfies K^{n-i} we can proceed by absurd. Indeed, if S does not satisfy the conditions

[7]Let us remember that the subscript used for the plan points out a portion of the plan starting from the position indicated until the end

expressed in K^{n-i} it means that either the first action of $[\pi_{n-i}]$ is not applicable or $S[a_i]$ does not satisfy $K^{n-(i-1)}$. The latter is not possible for inductive assumption as $K^{n-(i-1)}$ is assumed to be a kernel, while for the former it is obviously impossible since if the action is not applicable then $S[\pi_{n-i}]$ will not satisfy the goal. This proves the contradiction.

□

8.3.3 Time Complexity

This subsection analyzes the time complexity for the *numeric kernel* construction even if it is worth noting that the kernel's computation can be done just once, so that it can be seen as a form of off-line processing. By observing the algorithm, it is quite evident that the main task to be performed is the iteration all along the actions involved in the plan. However, each step consists of a number of substitutions, which in turns depends on the number of the comparisons and assigners given as input to the substitution. That is, it depends on the kernel previously computed.

Therefore let n be $|\pi|$, an upper bound on the total number of steps to be performed corresponds to $O(\sum_{i=0}^{n} m_i)$ where m_i is the number of substitutions involved at step i. Let us note that, as the iteration goes on, the number of substitution monotonically increases. In particular at the last step of the computation, the number of substitutions to perform is equal to the sum of the comparisons previously encountered. Therefore, $\sum_{i=0}^{n} m_i \leq \sum_{i=0}^{n}(iM)$ where M is the maximum number of comparisons among the numeric preconditions of the actions and the goal G.

It follows that the number of steps of the algorithm is given by $\sum_{i=0}^{n}(iM) = \frac{n(n+1)}{2}M$. Meaning that, asymptotically, the *numeric kernel* time complexity is $O(n^2M)$.

As reported in the previous example, given two comparison sets, A and B, the set resulting from their union is not always equal to the sum $|A| + |B|$ of their elements. Indeed it may be the case that at least a comparison is stronger than another comparison and hence the latter is useless and can be removed. The number of the elements to be analyzed during the substitution mechanism can actually be less, thus improving the performance of the kernel construction.

8.4 Improving the FLEX-RR Supervision Task

As we have seen in Chapter 6, the execution of the plan in the real world could fail because of the occurrence of exogenous events as well as unexpected action behaviors; therefore a continuous monitoring is needed.

8.4.1 Improved Monitoring

The FLEX-RR monitoring task consists in anticipating each action execution with a phase aimed at assessing the feasibility of the plan as a whole. Indeed in a plan execution monitoring context, the only checking of the action preconditions may in general not be enough; even if the action can be applicable, the executability of the remaining plan could be feasible only if certain conditions hold. To capture such conditions, FLEX-RR individuates three levels of plan consistency, where we have that a plan can be either *invalid*, *partially valid* or *valid*.

In the FLEX-RR agent, such a verification requires a simulation step, which is accomplished by the *Predictor* module. That is, given the plan of actions to be executed, the agent can predict the final state of the system by simulating the plan execution according to the model of actions and the observed current state of the system. Once such a final state is computed, the agent evaluates the prediction against the requirements expressed in the goal statement and infers the executability status of its plan.

Of course by exploiting the kernel formulation[8] the agent can avoid this simulation step.

In particular, it suffices to combine the plan consistency notion with the kernel's definition as the following:

Definition 17. *Given an MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, and a plan* $\pi = \{a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})\}$ *solving* Π*, let* S^i *be the* observed *system state obtained after the execution of the first i actions in* π*, and a set of kernels* $K = \{K^0, K^1, ..., K^n\}$*,we say that:*

- π *is* valid *at execution step i iff* S^i *satisfies both* K^i_p *and* K^i_{num}
- π *is* partially valid *at step i, iff* S^i *satisfies* K^i_{prop} *but not* K^i_{num}*;*
- π *is* invalid*, iff* S^i *does not satisfy* K^i_{prop}*.*

In other words, the task of monitoring can be performed by matching the state against K^i_{prop} and K^i_{num}. Indeed, the new formulation of the plan validity does not depend on the action still to be performed and the agent can focus just on the relevant numeric information.

The extension in the FLEX-RR architecture is straightforward; the Predictor module does not require the information from the MMA plan, rather it can combine the information of the state with the Kernel's structure. Of course, when the plan undergoes some adjustments, the kernel's set has to be recomputed consequently. In case of a changing in the modalities (i.e. when just *Recon* is applied) the only structure to be recomputed is the *numeric kernel*.

[8] It is important to remember that FLEX-RR has to be able to handle plans involving both propositional and numeric aspects, for which both the propositional and the numeric kenrel are necessaries.

8.4.2 Safe Control Delegation

Section 3 defines that an action is applicable in a state S when both the propositional preconditions and *at least* a modality of execution is satisfied in S.

By exploiting the kernel for a given MMA plan, we can in principle have multiple ways of applying an action without compromising the executability of the same plan. Besides the action applicability we can indeed introduce the concept of *safe execution*. That is:

Definition 18 (Safe execution)**.** *Given an MMPP* $\Pi =< A, I, G_{prop}, G_{num} >$*, a plan* $\pi = \{a_0(m_0), a_1(m_1), .., a_{n-1}(m_{n-1})\}$ *solving* Π *and an ordered set of numeric Kernels* K_{num} *for* π *and* G_{num}*, a safe execution of* a_0 *in a state* S *is a subset of modalities* M *of* a_0*, where each* $m \in M$ *is such that:*

- *the numeric precondition for* m *are satisfied in* I_n
- $S[\{a_0(m)\}]$ *satisfies the condition in* K_1

We can also extend the concept for the rest portion of the plan. That is, let S^i be the state observed, the safe execution for the i-th action is the set of modalities that guarantees the rest of the plan to be valid starting from S^i.

One of the most interesting positive effects of the safe execution notion is that, during the execution and given the actual state observed, it is possible to identify a set of modalities for the next action to be executed by matching a single step of simulation with the upcoming kernel.

The algorithm 9 implements such an intuition. The procedure takes in input the action a to execute, the state in which a has to be executed and the *numeric kernel* of the next part of the plan. For each modality of a the algorithm verifies that the numeric part of the state resulting from the action application (in that modality) satisfies the kernel's condition. If it is the case the modality is added to set of the safe execution modalities.

It is worth noting that the output M is such that $|M| \geq 1$. A different scenario would mean that the plan to be executed is not valid anymore. To avoid such a situation it is necessary that the function is called before the plan has been declared consistent.

By combining the notion of safe execution modality with the plan consistency Definition 10 reported in Chapter 4, it is possible to observe that:

Observation 7. *Let* S^i *be the current observed state of the system and let* $\pi = \{a_0(m_0), ..., a_i(m_i), ..., a_{n-1}(m_{n-1})\}$ *be valid at step i, if* m_i *belongs to the set of safe execution modality for* a_i*, then* π *will be valid at step i+1 according to* $S^i[a_i(m_i)]$*, too.*

The observation above indicates that if a plan is valid at some point of the execution and we apply an action with a safe execution modality then the plan will be valid also in the next step.

Algorithm 9: Safe Execution Modalities

Input: a - action;
S - state;
k - numeric kernel
Output: M - the safe execution modalities set

```
1 M = {};
2 foreach mod ∈ Mod(a) do
3     S' = predict(S,a,mod);
4     if S'_num satisfies k then
5         M = M ∪ {mod}
6 return M
```

The set of safe execution modalities includes those action modalities that not compromise the feasibility of the plan. For this reason they give a further level of flexibility to our system. In particular they establish the set of modalities that can be delegated to a further control mechanism. In Chapter 9, we will show an action supervision which guide the control over the action, by exploiting the *safe execution* notion presented so far.

8.4.3 Kernel Replanning

In the previous Chapters, we have seen that a plan may be repaired in the space of modalities by means of a reconfigurator; that is *Recon* adopted by FLEX-RR. Moreover, another option refers to repair the plan directly via a replanning from scratch. The reconfiguration mechanism allows the agent to focus only on a portion of the problem, trading flexibility for efficiency. On the other hand, the Replanning from scratch mechanism provides a further level of reasoning to achieve an high degree of flexibility to be used just in case.

As we observed in the experimental session, however, the replanning from scratch may be too costly from a computation point of view. For this reason in this subsection we propose a very simple strategy which directly applies the notion of the kernel.

Indeed, as well as the kernel allows the agent to understand when interrupting the execution, the same kernel, by definition, individuates the conditions to (re)-establish the applicability of the current plan.

Therefore, the notion of the kernel seems quite appropriate in the context of *repair* as it captures exactly the requirements for going back to the old situation. That is, given the expected kernel k and the current state of the system S, instead of searching for a solution satisfying the goal of the original problem, it could be sufficient to steer the search towards the expected kernel. Then, the "patch"[9] produced (if any) will bring the agent into a new situation

[9]The "patch" is actually an MMA plan as the original one. We used the "patch" term just

satisfying the kernel, meaning that, once the agent will execute the actions involved in this "patch", the original plan will be newly executable. We call this strategy Kernel Replanning.

The *Kernel Replanning* can be triggered by any continual planning agent (e.g. FLEX-RR) whenever the plan is not valid. Let S be the current state, π the plan to be executed, and K the kernel for π and a goal G, the main steps to integrate are:

- compute a patch ω to achieve K from S;
- *if* ω exists *then* append π to ω ; /* KERNEL-REPLANNING*/
- *else* substitute π with a new plan from S to G /*REPLANNING*/
- *if* π is not empty continue the execution otherwise abort

From the point of view of FLEX-RR it is worth remembering that the strategy invokes a replanner whether:

- the plan is invalid
- the plan is partially valid and the reconfiguration failed

Therefore the kernel re-planning can anticipate each replanning phase with the step reported above. More precisely, for each step i of the execution which requires the replan, it suffices to invoke the replanner tool as reported in Chapter 6 by replacing the goal conditions with the i-th kernel expected.

As experimental session we tested the mechanism for a particular domain, which is the *ZenoTravel* Domain. The next Section reports some interesting results, which proves the promise of the approach.

8.5 Experiments

This Section reports some experimental results aimed at verifying the validity of the kernels and in particular the Kernel Replanning presented in Section 8.4.3. We conducted the experiments in the *ZenoTravel* domain. For the purpose of over-stressing the kernel based replanning mechanism, we considered action with just one modality of execution. The resulting domain definition corresponds to the original version of the domain proposed in the Third International Planning Competition (see `http://www.dur.ac.uk/d.p.long/competition.html`).

One of the attractive characteristics of the *ZenoTravel* domain for our purposes is that the fuel of the planes is a renewable and continuous resource,

to disambiguate the original solution from such an intermediate course of actions, whose only aim is to re-establish the previous conditions

indeed, a plane can refill its tank once there is no more fuel at disposal. However, the original problem definition does not impose where and when the refuel is possible.

To make this domain more challenging, we defined an extended version in which refuel is possible only in certain city. To this end, we modified the preconditions of the refuel action for allowing its execution only in cities where it is explicitly stated the presence of the refuel station. This means that a plane should pay more attention in choosing the paths for moving people all around. Moreover, this causes a new interdependence between the propositional aspects (the presence of the refuel station can be modeled in a propositional way) and the numeric ones (the fuel is modeled as a numeric fluent).

We performed experiments in both the original and the extended version of the domain, which we will call the *normal* and *hard* domain. The generation of the cases has been done starting from the suite of problems computed for the planning competition. In particular, we focused on the 7 most difficult cases as they are the most interesting for the repair problem. Vases differ among each other on the number of people, cities and planes to be taken into account; our tests suite involves problems with people ranging from 10 to 20, cities from 5 to 20 and at most 5 planes.

To validate the monitoring and the repair via kernel, the tests refer to the execution of such plans in not nominal situations; we injected discrepancies in the way in which the fuel is consumed throughout the plan execution. For each case we injected 5 different amounts of noise, which consequently produced 5 different instances of repair problems. The total number of experiments is hence 35 cases for each domain. The experimental software architecture is the same reported in Chapter 7.

Tests ran on two different configurations of FLEX-RR. The one with the only replanning from scratch methodology (FLEX-REPLAN), while the other one with the replanning replaced with the Kernel Replanning[10]. For each case we measure (i) the performance of the system in terms of cpu-time spent for solving the planning task, and (ii) the quality of the solution obtained, that is, the number of actions in the new plan. For the replanning case, the new plan is the set of actions from the current state towards the goal. For the Kernel Replanning case, the new plan is the concatenation of the actions obtained for reaching the expected kernel with the old plan.

[10]In the domain definition we used just a modality for each action. For this reason the reconfiguration mechanism is useless in this formulation. As a future work, it may be interesting to assess the system in the context of many modalities of execution where a comparison with the reconfiguration mechanism will be possible

8.5.1 Performance::Cpu Time

The Figures 8.1 and 8.2 summarize the performance results for the two different strategies; that is, they measure the cpu-time (in msec) spent for the resolution of the planning task. The x-axis enumerates the 7 cases considered whereas the y-axis the computational effort. The one (red line) is the Kenrnel replanning strategy (we turned off the replanning from scratch), the other one (blue line) is the replanning from scratch methodology (we turned off the repair mechanism). Both strategies, as anticipated, have been experimented for the *normal* and *hard* domain (8.1 and 8.2 respectively).

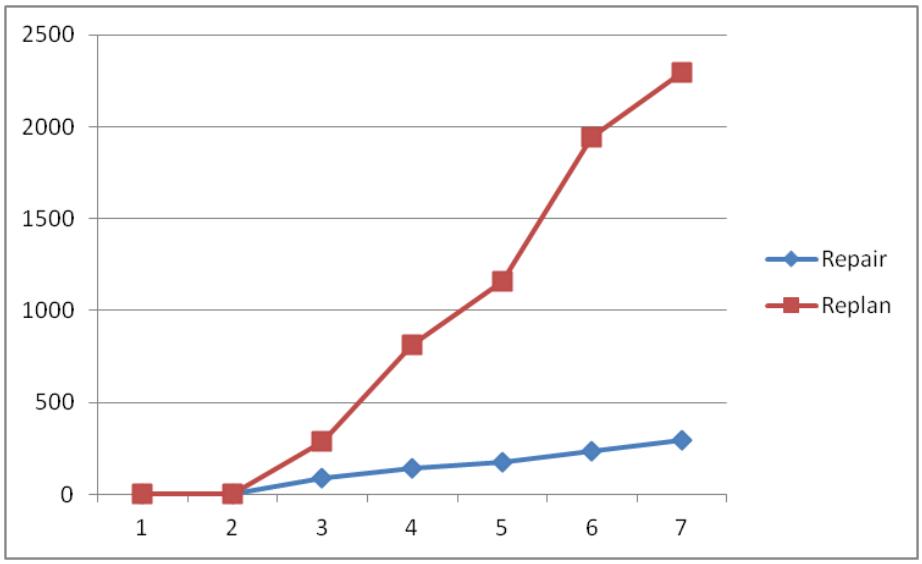


Figure 8.1: ZenoTravel Domain :: Normal Version :: Cpu-time

It is quite clear that for this class of problems, the repair mechanism outperforms the replanning from scratch in all cases we tested, and it is true even when dealing with hard cases. In this latter cases the replanning performs rather bad since the performance degrades rapidly. For instance in case number 6, the replanning from scratch found a solution only after 25 secs.

One of the interesting property showed by the proposed Kernel Replanning mechanism, is that in all test cases the amount of the time spent for re-establishing the condition is always under 1 sec.; as long as the replanning mechanism becomes more and more time expensive, the repair remained stable. This result is very surprising; in fact the planner employed (*Metric-FF*, [50]) turns out to be very good in dealing with the repair problem at hand.

In all test cases, the monitoring devoted to intercept the plan inconsistency has been performed by checking the conditions in the kernels set (both for the replanning and the kernel replanning strategies). The cpu-time spent turned

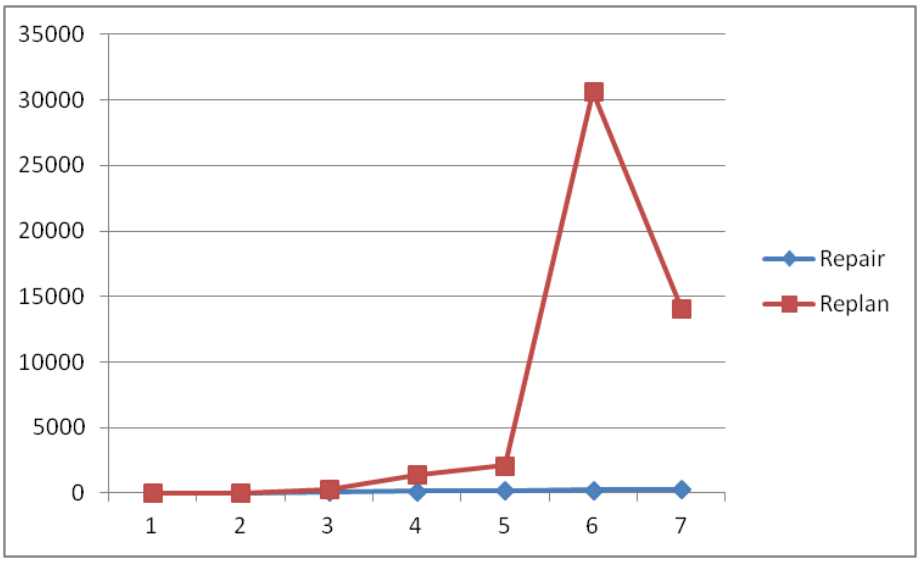


Figure 8.2: ZenoTravel Domain :: Hard Version :: Cpu-time

out to be negligible as the number of comparisons for the checking problem has been always less than 10-20 comparisons.

8.5.2 Quality::Plan Length

Figures 8.3 and 8.4 report the average length of the plans for each class of difficulty tested. Thus, we measure the average length from the 5 different solutions obtained.

As for the Cpu-time benchmark above, the x-axis figures out the case instance, while the y-axis reports the number of actions, i.e. the length of the plan computed to reach the goal[11].

Surprisingly, the average length of the plan measured turns out very similar for both architectures. In the hard domain we have noted that repair performed even better that replanning from scratch. The result is very promising, even unexpected, since the repair mechanism can just try to put a patch on the plan. This means that it may be the case that it performs actions with the only scope to repair the current flaw (e.g. the fuel is not sufficient for the current plan) without an interleaved and optimized integration with the remaining part of the plan. On the other hand instead, the replanning from scratch is allowed to explore all the possible plans to reach the goals, which should have given it the chance of producing more optimized plan. However, this was not the case.

[11] Let us remember that for the repair case this plan involves both the bridge and the old plan

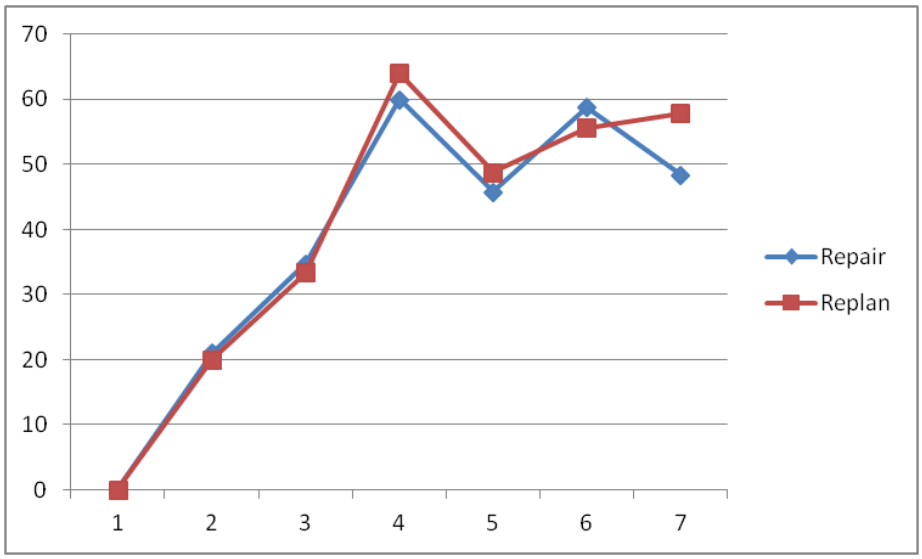


Figure 8.3: ZenoTravel Domain :: Normal Version :: Length of the Plan

8.6 Conclusions

This Chapter proposed an extension of the propositional kernel ([40],[43]) called *numeric kernel*, which is specifically designed for dealing with actions involving changes for continuous variables, i.e. numeric fluents. The *numeric kernel* accompanied to the well known propositional kernel, allows to improve the monitoring throughout the plan execution guaranteeing the agent to focus only on the relevant part of the current state.

Concretely a *numeric kernel* is expressed as a set comparisons which must be satisfied in order to make applicable a plan of actions. Therefore, given a state of the system, the monitoring problem can be performed without any propagation and any search in the plan to be performed.

Moreover, it is worth noting that complex plans may involve deep interactions among numeric fluents. Understanding which are the relevant information for the assessment of the plan validity may be an hard task. The *numeric kernel* hides such a complexity, by allowing to focus only on a subset of such variables, which are the ones actually relevant for the problem at hand.

Exploiting the *numeric kernel* notion, the Chapter reported three main enhancements for FLEX-RR:

1. **Improved Monitoring**, a focused way for performing the assessment of the validity conditions for a plan;

2. **Kernel Replanning**, an alternative replanning strategy which is based on reestablishing the nominal conditions without making a blind replan-

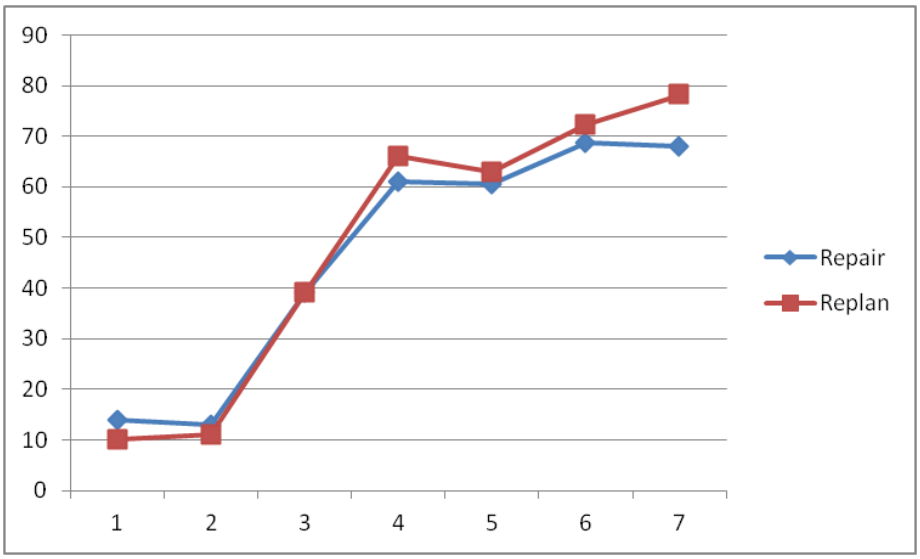


Figure 8.4: ZenoTravel Domain :: Hard Version :: Length of the Plan

ning from scratch

3. **Safe Control Delegation**, a mechanism to identify *safe execution modalities*, which is a set of modalities that can be applied in a given status without compromising the feasibility of the plan. In particular in Chapter 9, we will see an Action Supervision module (ACTS) that can exploit such a notion to integrate its *local* control with the plan supervision task performed by FLEX-RR.

To validate the notion, we applied the **Kernel Replanning** in the *ZenoTravel* Domain. Results showed that, when the repair faces unexpected resources consumption, a focused repair works very well w.r.t. a replanning mechanism. Moreover the length of the resulting plan, surprisingly, turned out to be on average better than the length of the plan produced by the replanning mechanism, even if the repair mechanism is not allowed to perform an exhaustive search into the plans space.

In many scenarios the numeric fluents model resources and it may be the case that a resource is consumable but not renewable (e.g. the time). For this reason, if the external events cause a greater resource consumption (for example a time delay), it is impossible to come back to the expected state (it is in fact not possible to come back in the time). Therefore the mechanism can be useful in some domain while it can be useless in other ones. Understanding under which conditions it is suitable the kernel replanning mechanism is one of our lines of research.

The notion of *numeric kernel* is quite general in the context of numeric

planning and we believe it could be investigated in a larger perspective of case-based-planning. However, as an immediate future work we would like to test the repair strategy we developed on a larger set of domains, to understand the generality of the repair approach. Moreover, we would like to study a more sophisticated way in the selection of the kernels towards which perform the patch; in this first version in fact we employ a conservative setting where the repair is performed directly towards the state that was expected.

Chapter 9

ActS:: Action Supervision through Multi Modality Actions

In this Chapter we propose a methodology for supervising durative actions while they are still in execution. Such a level of control complements the role of FLEX-RR in that the supervision is performed at the level of the action instead of the plan. The approach is tested on a specific domain, which is the *Planetary Rover* domain reported throughout the thesis.

9.1 Introduction

As we have seen in the previous chapters, the strategy of robust execution we proposed is able to control the way in which actions have to be executed all along the plan execution, but just *before* their actual execution.

As soon as some kind of plan inconsistency arises, FLEX-RR invokes a repair strategy, which in extreme situations may become a re-planning step. These methodologies, however, are unable to intervene during the execution of an action, as the repair is invoked just after the detection of a *plan inconsistency*, that is, when the plan execution has been interrupted.

Actually, FLEX-RR mitigates this problem by anticipating the configuration of future actions in the plan. As a matter of fact, FLEX-RR behavior does not depend directly on a failure of an action, rather it predicts the behavior of the system for the whole execution. For this reason the repair strategy can be invoked earlier, meaning before we assess that an action with a specific modality cannot be executed anymore. Unfortunately, it is not always possible to anticipate the detection of threatened actions, since the plan executor (i.e.,

the agent) may have just a partial knowledge of the world where it is operating; this imposes the granularity of the control to focus just at the plan level. That is FLEX-RR has no control during the action execution.

In this Chapter we complement the repair mechanism proposed in Chapter 6, with a new module, namely Active Supervisor (ActS). The main objective of ActS is to avoid (at least in some cases) the occurrence of *action* failures *during* the same action execution improving as a consequence the robustness of the whole plan execution. More precisely to avoid failures ActS aims at assuring that (i) the action execution does not prevent the safety of the system, (ii) the action achieves the expected effects (including the way in which resources are employed). For these reasons, ACTS plays the role of a short term supervisor, while FLEX-RR of a long term one.

To reach its purposes, ActS is up to adapt the way an action is carried on to the current contextual conditions; that is, the mechanism exploits the fact that each action in the plan is associated with a set of execution modalities that embodies alternative ways to reach the action's effects. Intuitively, an execution modality in this context can be seen as a specific configuration of the rover's devices. When an action is submitted for the execution, the initial modality can be adjusted on-line by ActS while the action is still in progress in order to adjust the rover's behavior without interrupting the execution phase.

ActS is a complex system involving a number of modules; in particular, it exploits a temporal interpretation module for monitoring the execution of the action and for detecting potential deviations from the nominal expected behavior over a time window. When potentially hazardous situations are detected, another module, called Active Controller, can decide to adjust the current execution modality by taking into account the suitability of alternative modalities in alleviating the discrepancy between the actual behavior and the nominal one.

The combination of FLEX-RR and ActS results in an hybrid control architecture, that is, deliberation (FLEX-RR) and reaction (ACTS) are achieved in a single framework. As shown in the variety of approaches reported in literature (the Alami's architecture, [5] , the CLARATY subsystem ,[76] and so forth), the trade-off between deliberation and reaction still remains an open problem when it comes to deal with real-world environments (e.g., mobile robot). Hybrid approaches represent the most suitable compromise when it comes to deal with real-world environment (e.g. mobile robots).

We have tested ActS in a specific domain, which is the *Planetary Rover* domain introduced in Chapter 1. Since the synthesis of a mission plan is a complex task, the rover is typically not allowed to significantly deviate from it; thus, in case of action failures, the rover can just interrupt the execution and wait for instructions (i.e., a new mission plan) from a Ground Control Station (GCS) on Earth.

To this end ActS re-configures the action without endangering the *structure* of the plan, and it changes only one action modality per time whereas FLEX-R trades part of the stability for flexibility. Indeed, once the reconfiguration is called in FLEX-R, more action modalities can be substituted all along the part of the plan still to be executed. Moreover, exploiting the notion of *Safe Execution Modalities* reported in Definition 18, ACTS has a means to prefer only the modalities which are supposed to not threat the whole plan consistency. Thus the integration between ActS and FLEX-RR can be more effective.

The Chapter is organized as follows: Section 9.2 recalls the space exploration scenario presented in Chapter 3; Section 9.3 presents the knowledge provided to ActS to perform its job; Section; Section 9.4 explains how the ActS is combined with FLEX-RR. Section 9.6 reports some experimental results that will evaluate the ActS contribution.

9.2 An exemplifying scenario: The Planetary Rover Domain - part 2

In Chapter 3, a space exploration scenario involving a planetary rover in charge of moving all along the surface of the planet and collecting information of the environment has been introduced. As we have seen, this scenario presents some challenging characteristics that made it particularly interesting for the plan execution problem. The rover, in fact, has to operate in a hazardous and not fully observable environment where a number of unpredictable events may occur.

It is easy to see that some of these actions can be considered atomic (e.g., take picture), some others, instead, will take time to be completed. For instance, a drive action will take several minutes (or hours), and during its execution the rover moves over a rough terrain with holes, rocks, slopes. The safeness of the rover could be threatened by too deep holes or too steep slopes since some physical limits of the rover cannot be exceeded. In case such a situation occurs, the rover will be unable to complete the action. Of course, the rover's physical limits are taken into account during the synthesis of the mission plan, and regions presenting potential threats are excluded *a priori*.

However, the safeness of the rover could also be threatened by terrain characteristics which can hardly be anticipated. For instance, a terrain full of shallow holes may cause high-frequency vibrations on the rover, and if these vibrations last for a while they may endanger some of the rover's devices. This kind of threat is difficult to anticipate from the planning point of view both because satellite maps cannot capture all terrain details, and because this threat depends on the rover's contextual conditions, such as its speed.

Since many approximations in the action model are mandatory to make the planning (and the reconfiguration) phase feasible, the anticipation of many

situations are deliberately postponed to the execution phase. For this reason, some actions should be supervised all along their execution. To this end, in the scenario of the *Planetary Rover* we extend the drive action to be considered as durative and hence controllable during its execution. The set of modalities to be considered remains the same introduced in Chapter 3 and reported in the Appendix C. That is, the drive action can be executed in one of three different modalities, i.e. "safe", "cruise" and "agile", that are characterized by three different speed configurations. Intuitively indeed, slowing down the rover's speed can mitigate the harmful effects of disconnected terrains. In such a scenario it is hence desirable to have a system that is able to (1) detect anomalous execution and (2) react by changing on the fly the action modality for the purpose of preventing future failures.

Moreover, as we will see the solution we propose is sufficiently flexible to change the execution modality not only when threats have been detected, but also when threats terminate and nominal execution modalities can be restored.

9.3 Knowledge for the active control

As anticipated in the introduction, the aim of this level of control is to give a reactive response in facing unpredicted contingencies while the action is still under execution.

This kind of behavior is actually performed by a module, namely Acts. Before discussing the internal architecture of ActS and how it intervenes during the plan execution phase, it is essential to introduce the pieces of knowledge ActS requires to carry on its job. The control at such a level of abstraction requires additional pieces of information which cannot be extrapolated directly from the action model introduced in Definition 3.

The first issue to face is about the representation of the plan.

Plan. The plan that is managed by the system is an MMA plan where each action is defined according to Definition 3. However, to make explicit the fact that action in this approach cannot be considered atomic anymore, we extend the MMA action model for directly expressing the invariant condition construct. The extension is inspired by the durative actions provided by PDDL 2.1 [42] which, besides preconditions and effects, allows the definition of invariant conditions (by means of the over all construct) that the planner must guarantee to maintain during the synthesis of the mission plan. These invariant conditions are exploited in the context of planning to enhance planners in discovering mutex relations for plans with concurrent actions. In our approach the invariant conditions are exploited during the execution of an action to check whether the rover's safeness conditions are maintained. For instance, the "over-all" construct of the drive action shown in Figure 9.1 specifies which conditions on the rover's

attitude (i.e., the combination of pitch and roll) are to be considered safe, and hence must hold during the whole execution of the action.

In a nutshell, the `condition` clause encodes the physical limits within which the rover's behavior is kept cautious while a drive action is in progress. Let us suppose that two parameters are essential for the safeness of a navigate action: `roll` and `pitch`. When either the absolute, or the derivate value of one of these two parameters exceeds a predefined threshold, the navigate action must be considered failed. In that case, ActS aborts the action in order to prevent further damages.

From the previous example it is apparent that an invariant condition is a Boolean statement, encoding when a failure situation occurs. When it is violated at some step of execution, it is too late for intervening: the plan execution phase must be interrupted and a new plan must be requested. On the contrary, we aim at preventing the occurrence of action failures by anticipating the violation of invariant conditions during the action execution. To do so we need some further pieces of knowledge associated with each action in the mission plan.

```
(:action drive
  :parameters (r1 l1 l2)
  :modalities (safe, normal, fast)
  :precondition (and (reachable l1 l2)(in r1 l1))
      (safe: ((>= (power r1) (f_pwr_safe(l1, l2)))
      (normal: ((>= (power r1) (f_pwr_normal(l1,l2)))
      (agile: ((>= (power r1) (f_pwr_agile(l1, l2))) )
  :condition (over all (and (<= (pitch-derivative r1) 5)
                            (<= (roll-derivative r1) 5)
                            (<= (pitch r1) 30)
                            (<= (roll r1) 30)))
  :effect (and
      (in r1 l2)
      (not(in r1 l1)))
      (safe:
         (decrease (power r1)(f_pwr_safe(l1, l2)))
         (increase (time) (f_time_safe(l1, l2))) )
      (normal:
         (decrease (power r1)(f_pwr_normal(l1, l2)))
         (increase (time) (f_time_normal(l1, l2))))
      (agile:
         (decrease (power r1)(f_pwr_agile(l1, l2)))
         (increase (time) (f_time_agile(l1, l2))) )
  ))
```

Figure 9.1: An MMA drive equipped with over all conditions

Execution Modalities for Acts. The concept of execution modalities that is defined in 3 is key in the ActS task. However, differently from the use made by

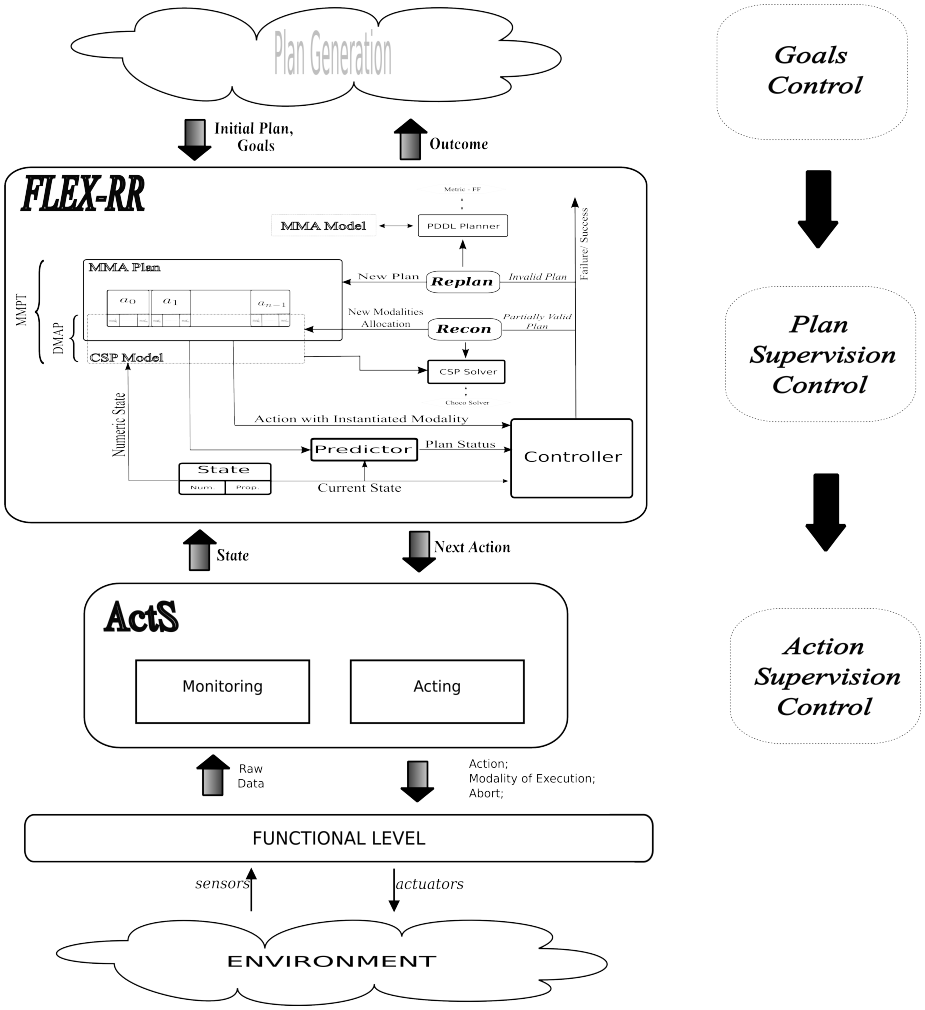


Figure 9.2: The FLEX-RR-ACTS Architecture

FLEX-RR, the execution modality here are not exploited for prediction reasons; instead they express in a discrete way those decisions that can be asynchronously performed over the rover's parameters. An execution modality can be seen as a specific configuration of parameters or devices to be activated that has an impact on the current behavior of the rover. In other words, given the set of modalities $mods(a)$ associated with a, each execution modality $m \in mods(a)$ represents a possible setting of the rover parameters (e.g. the rover speed during the navigation can be set to 1 m/s as well as 2 m/s).

For instance, similarly to the model adopted for FLEX-RR, the drive action can be associated with three alternative modalities: $mods(\texttt{drive})$={*cruise*, *safe*, *agile*}. In our experiments, we focus on the speed parameters[1]. In such a

[1]In general the safety of a drive action may depend on other parameters such as maxi-

```
chronicle hazardous-terrain {
 event(medium-hazard[pitch, roll], t1 )
 event(medium-hazard[pitch, roll], t2 )
 event(severe-hazard[pitch, roll], t3)
 t1<t2<t3 ; t2-t1<W1 ; t3-t2<W1
 when recognized { emit
   event(hazardous-terrain[pitch,roll],t)} }
```

Figure 9.3: A chronicle recognizing a hazardous terrain.

```
chronicle plain-terrain {
  occurs( (N, +oo), no-hazard[pitch, roll], (t, t+W) )
  when recognized {
    emit event(plain-terrain[pitch,roll],t);
  } }
```

Figure 9.4: A chronicle recognizing a plain terrain.

perspective, the *cruise* sets the nominal speed of the rover, while *safe* and *agile* are alternative modalities obtained by decreasing or increasing, respectively, the nominal speed.

This means that, *during* the execution of a drive action, ActS can adjust the rover's velocity; this is important as the derivate values of `roll` and `pitch` will strongly depend on the actual speed of the rover; by tuning the speed parameter, ActS can therefore prevent a failure of a drive action. For expressing the relation among particular trends of execution and action modalities we need to reason over the time, i.e. we need to understand under which conditions a situation harms the safeness of the rover's equipment.

Temporal Patterns of Behavior. To prevent the occurrence of an action failure, it is essential to recognize anomalous trends of execution while the action is still in progress. Each action is therefore associated with a set of temporal patterns, each of which describes a sequence of events that should, or should not, occur during the nominal execution of an action. While many other formalism can be employed, in this work we adopted the chronicles formalism [39] to encode these temporal patterns. Intuitively, a chronicle is a set of events linked with each other by temporal constraints. Since they allow to model the behavior of dynamic systems over time, chronicles have been successfully exploited for real-time monitoring and diagnosis of Discrete Event Systems (DES), even of large dimensions as telecommunication networks (see e.g., [79]).

In our case, the events that we want to capture within a chronicle correspond to relevant changes in the status of the rover which indicate potentially anomalous behaviors. These events may depend both on the activities carried on by the rover itself, and on the contextual conditions of the environment where the

mum/minimum distance from the obstacles

rover is operating. Thus, each durative action $a \in P$ is also associated with a set $chrons(a) = \{chr_1, chr_2, \ldots, chr_m\}$ of chronicles, where each chr_i represents a trend of execution, either nominal or anomalous, that is relevant for recognizing what is happening during the execution of action a.

Figures 9.3 and 9.4 show two examples of chronicle associated with a navigate action. The first chronicle identifies a potentially hazardous terrain. This chronicle is recognized when at least N *severe-hazard* events have been detected within an interval of W time instants. The basic idea is that the safeness of the rover may be endangered when it moves at a high speed along a too rough terrain; this kind of threat can be captured by detecting hazardous variations of the roll and pitch parameters in a short time window. The second chronicle recognizes a plain terrain when for a period of at least N time instants the rover status remains nominal.

It is important to note that, to keep the definition of chronicles a simple task, they mention high-level events such as *medium-hazard* and *no-hazard*; these events result from an interpretation process over the rover's status variables. In particular, it is easy to see that, since the execution of a drive action is aborted when the rover's roll and pitch exceed predefined thresholds, these two parameters play an important role during the process of generating the high-level events mentioned within the chronicles in $chrons(\texttt{drive})$. We will describe the interpretation process in the next section.

9.4 Action Supervision

As anticipated in the introduction the objective of ActS is to complement the task performed by FLEX-RR by providing a further level of symbolic control before the delivery of actions to the functional devices.

The architecture reported in Figure 9.2 highlights hence how ActS interacts with the rest of the environment. In particular ActS interacts

- (on the top), with FLEX-RR for receiving the next action to execute and providing (to FLEX-RR) the new system status once the action is terminated.

- (on the bottom), with the functional level for receiving the raw-data from the sensors (or a first abstraction of them) and for submitting the action to execute, as well as to change on the fly the modality while the action is still in execution.

The architecture makes clearly evidence of the two different levels of control. The former focused on the Plan Supervision task (FLEX-RR) and the latter on the Action Supervision task (ACTS).

Besides the simple action start message and the modality submission, as explained before, ActS can also send an abort message. The message allows the system to interrupt the action execution in case particular safety conditions are violated. It is worth remarking that, while the abort is sent to the functional layer, a similar message is not sent to FLEX-RR. The reason why we prefer this strategy is because FLEX-RR has no chance for the reestablishment of the mission in such a situation. The abort message is in fact a very extreme decision, which may requires a new planning phase or even an abandonment of certain sets of goals. For this reason in our system we consider the abort as a failure outcome that must be issued towards the planner[2].

Let us start with the ActS's internal architecture.

9.4.1 ActS's Internal Architecture

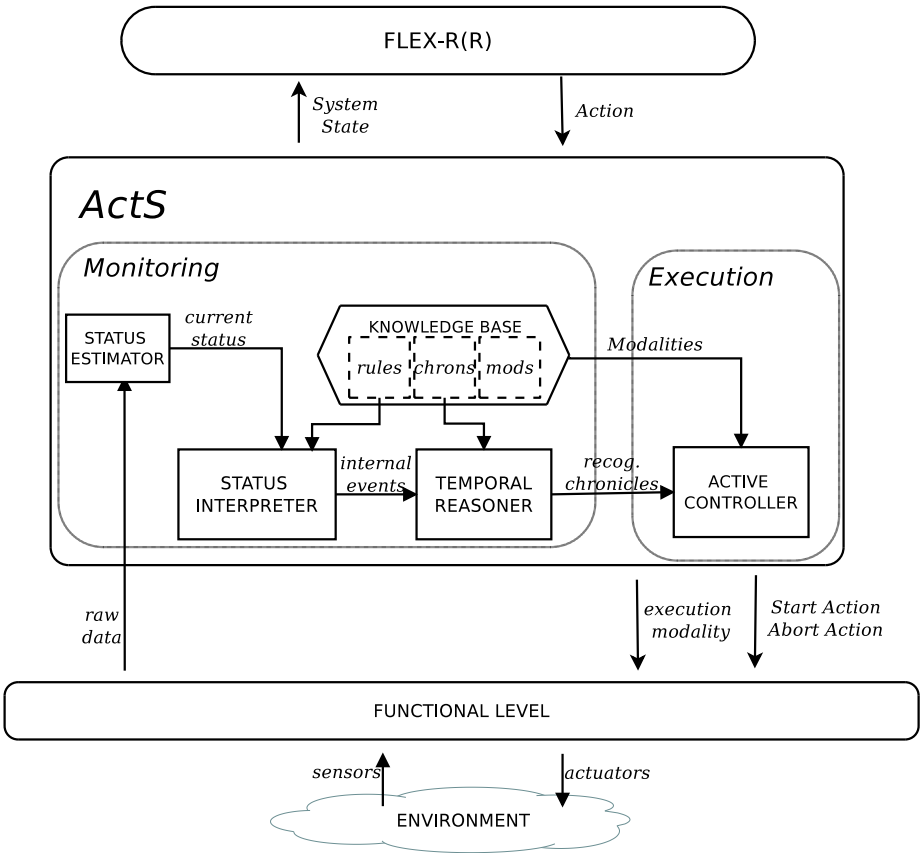


Figure 9.5: The ActS internal Architecture

[2]In space exploration system, the planning activities are not performed automatically; rather they are quite often a long process involving both humans and automated planners ([16])

The internal architecture of ActS involves a number of modules:

- The **Status Estimator** (SE) gets, at each time t, the raw data provided by the FL and produces an internal representation of the current rover's status. The result of the SE can be thought of as an array $status_t$ of status variables representing a rover's snapshot at time t. The SE can carry out qualitative abstractions of the collected data, therefore each variable $v \in status_t$ is set to a specific value which may be either a quantitative or qualitative value. The internal status $status_t$ produced by the SE is made available to the Status Interpreter of ActS. At the end of the action execution, the module provides also the Plan Supervisor with the last inferred status of the system.

- The **Knowledge-Base** (KB) maintains the pieces of knowledge exploited by ActS for detecting anomalous trends and for compensating them by tuning the execution modality of the current action. In particular, for each action type *type*, the KB maintains the set of execution modalities $mods(type)$ and chronicles $chrons(type)$ associated with it. The execution modalities are the ones defined in Chapter 3, Definition 3, enriched with specific parameter configurations (e.g., the devices to use). Moreover, the KB includes a set of interpretative rules that helps the Status Interpreter in its job.

- The **Status Interpreter** (SI) has to generate the (high-level) internal events that are mentioned within the chronicles. It exploits a set of interpretative rules associated with the current action. These interpretative rules have the form *Boolean condition* $\rightarrow$ *internal event*. This condition is built upon three basic types of atoms: status variables x_i, status variable derivates $\delta(x_i)$, and abstraction operators $qAbs(x_i, [t_l, t_u]) \rightarrow qVals$ which map the array of values assumed by x_i over the time interval $[t_l, t_u]$ into a set of qualitative values $qVals = \{qval_1, \ldots, qval_m\}$.

  For example, the following interpretative rule:

$$\{(\delta(roll) > limits_{roll} \vee\ \delta(pitch) > limits_{pitch})\}$$
$$\Downarrow$$
$$severe\text{-}hazard(roll, pitch)$$

  is used to generate a *severe-hazard* event whenever the derivate value of either roll or pitch exceeds predefined thresholds in the current rover status.

  Another example is the rule:

$$\{(attitude(roll, [t_{current} - \Delta, t_{current}]) = nominal)$$
$$\bigwedge$$

$$(attitude(pitch, [t_{current} - \Delta, t_{current}]) = nominal)\}$$
$$\Downarrow$$
$$safe(roll, pitch)$$

where *attitude* is an operator which abstracts the last Δ values of either roll or pitch (the only two variables for which this operator is defined) over the set {nominal, border, non-nominal}.

- The **Temporal Reasoner** (TR) is essentially a Chronicle Recognition System (CRS) similar to the one proposed by Dousson in [39]. It is responsible for triggering the Active Controller once a new chronicle has been recognized.

- The **Active Controller** (AC) accomplishes two important activities. First, it selects an execution modality to be issued towards the FL. In principle, such a selection should correct the current robot's behavior smoothly; that is, on one side, the AC's strategy should not be too reactive in order to avoid abrupt changes in the robot's behavior which may be as dangerous as the threat to face; on the other side, the AC should be able to restore the nominal execution modalities when it is reasonable to presume that no menace is expected in the near future. Second, the AC updates some parameters of the current action according the execution modalities it emits. To make the selection of the modality more intelligent, AC is guided by the *Safe Execution Modalities* computed by the extension of FLEX-RR proposed in 8.

9.4.2 Algorithms

In the previous subsection we have introduced the main modules of ActS; in this subsection we discuss how these modules are actually integrated with each other to provide the supervision service ActS is responsible for. The main supervision task is reported in algorithm 10, whereas algorithm 11 explains the transduction process which starts by polling the Raw Data from the FL and ends with the submission of qualitative events for the temporal reasoner module. Finally the algorithm 12 is in charge of deciding the actual modality of execution. It is worth observing that the last algorithm has to provide a prompt answer. For this reason, the decision is quite a direct consequence of the monitoring performed before.

The ActS supervision task is activated once the action is submitted to the Functional Layer. Actually the role of the action supervision makes sense when dealing with durative actions. For this reason, we discern actions in *controllable* actions (durative actions) and non controllable (i.e. *atomic* actions). Whether

Algorithm 10: ActiveSupervision

```
   Input: a - action, sem - safe_exeucution_modalities , c - modality
 1 submit a to FL;
 2 if a is controllable then
 3     t = 0;
 4     while a is not ended do
 5         <chronicles,result> = Monitoring(a,t);
 6         if result = abort then
 7             abort(a) command to FL;
 8             wait for abort completion from FL ;
 9         else
10             ActiveController(a,sem,chronicles);
11         t = t+1;
12 else
13     wait for completion from FL ;
```

a controllable action is submitted to the FL, the action supervision is activated; otherwise this simply waits for the *end* message provided by the FL. Let us introduce with more precision the active supervision procedure.

Active Supervision

The algorithm takes in input the next action to be executed,the *Safe Execution Modalities* and the initial modality associated with the action. As a first step the algorithm submits the action toward the FL with the current modality and checks if the action is actually controllable. As anticipated the supervision is possible only whether the action is *durative* and hence controllable.

If this is not the case the algorithm simply waits the acknowledgment of the FL for the action completion; otherwise the actual active supervision starts.

The supervision is performed at a predefined frequency and for each iteration the temporary variable t is increased by one thus keeping trace of each execution step[3].

For each iteration, the algorithm firstly invokes the monitoring routine then, depending on the outcome, can either perform the control on the action modality or return an abort.

In a nutshell, the *ActiveSupervision* is up to emit (when necessary) an execution modality towards the FL so that the trend of the current action can be corrected. This result is obtained by the cooperation of the *Monitoring* (*Status*

[3]The pseudo code reported above does not make any assumption on the update frequency. We have actually performed tests in 1hz. It is worth noting that the frequency of update actually depends on the amount of reasoning time needed for performing the whole supervision task and this strongly depend on the particular domains of application. However the strategy can be easily configured for different frequency of control

Algorithm 11: Monitoring

Input: a - action, t-time
Output: RC - chronicles , abort - message (just in case,otherwise *null*)

```
1  status_t = interpret_raw_data();
2  if status_t ⊭ invariant_condition(a) then
3      return <∅,abort>
4  H = append(H,status_t);
5  rules(a) = get_interpretative_rules(a);
6  events_t = status_interpreter(H,rules(a));
7  RC = ∅;
8  chronicles(a) = get_chronicles(a);
9  foreach event e ∈ events_t do
10     chr(a) = get_relevant_chronicles(a,chronicles(a));
11     if TemporalReasoner(chr(a),e) is recognized then
12         RC ∪ chr(a);
13 return <RC,null>
```

Algorithm 12: Active-Controller

Input: a - action, sem - safe_exeucution_modalities, rc -chronicles

```
1  mods(a) = get_associated_modality(rc);
2  modality= most_suitable_mod(sem,mods(a));
3  submit modality to FL;
```

Interpreter (SI) and *Temporal Reasoner* (TR)) and the *Active Controller* (AC).

Monitoring

The details of the monitoring task are reported in Algorithm 11. As a first step, the algorithm interprets the raw data provided by the Functional Layer. As explained above such a computation is performed by the SE. This gives the Active Supervision a first abstraction on the status of the system. Then the algorithm matches the inferred new state with the invariant conditions expressed in the model of the durative action. Whenever such invariant conditions are not satisfied the procedure will immediately return with an abort message toward the main routine, which in turns will issue the abort command towards the FL and will wait for its completion. The functional layer may take some time to perform the abort of the action; for this reason it is safe that the system will suspend the control until the abort is actually performed.

If the invariant conditions are satisfied, the action can proceed the execution; however, the situation requires a deeper analyses as also minor deviations from the nominal behavior may be the cause of future failures. This kind of reasoning is performed in a temporal dimension by the CRS module presented so far. More precisely, the recognized situation (i.e. the chronicles) are computed from line

9 to 12. Since the chronicles are activated on a set of events, these have to be extrapolated by analyzing the status history against the rules associated with the action currently in execution. This task is up to the SI, which exploits a history of rover's past states (H) together with a set of interpretative rules in order to synthesize the internal events representing relevant changes in the rover status: $events_t$ is the set of internal events generated by the SI at each time t. Each event $e \in events_t$ is subsequently sent to the TR.

For simplicity, in our approach we assume that each event e can be consumed by exactly one active chronicle chr; the function *get-relevant-chronicle* in Figure 11 selects such a chronicle from $chronicles(a)$ so that the TR receives in input just the event e which influences that chronicle. By consuming the events it receives from the SI, the TR will eventually recognize a chronicle chr_a. In general, more chronicles can be recognized simultaneously, so all the chronicles recognized at time t are collected into the set CHR.

Active-Controller

The task of the Active-Controller consists of two main steps (see Algorithm 12). Firstly, the algorithm extracts the modalities which have been recognized by the monitoring; then it selects from this set the next modality to be issued to the FL.

The function *most_suitable_mod* performs the choice on the basis of the *Safe Execution Modalities* set. In particular the set is used to establish what are the modalities that should be preferred. The ratio is to guide the selection towards those modalities which are supposed to not compromise the feasibility of the plan. As indeed observed in 7, the *Safe Execution* property guarantees that, whether other contingencies does not arise, if an action is applied with a *Safe Execution* in a given state of the system the rest of the plan remains valid after the action execution. Hence, FLEX-RR will not need any further computation.

However the rank in the current implementation only distinguishes two levels of application; for this reason there may be more than one modality with the best rank. At the moment we break ties by picking up the modality according to the order in which they appear in the MMA model.

There are two possible heuristics that can be employed in the selection mechanism reported above.

Domain Independent Heuristic. The idea is that, since action modality may differently affect the use of resources, one may guess the modality that *better* is supposed to improve the satisfaction of conditions expressed in the upcoming numeric kernels. For instance, let us consider a scenario where a drive action has been slowed down due to the presence of many vibrations. Once the vibrations have terminated, the monitoring recognizes that both cruise and fast modality

can be selected. However the next kernel imposes a very strict constraint on the time to the end of the action. So, even if both the cruise speed and the fast speed belong to the set of *Safe Execution Modalities*, the active control may choose to pick the fastest one as that represents a greater chance to satisfy the kernel constraint once the action is terminated.

Domain Dependent Heuristic. The idea is to provide further knowledge to the domain. For instance one could imagine to have another rank among modalities in such a way that given a set of possible applicable modalities, the active control would pick the one *closer* to the current execution one. Such a kind of approach would be aimed at minimizing the impact on the change of modality since presumably, closer modality could request less impact on the parameter configuration (or the use of different devices) w.r.t. distant modalities.

Both heuristics are under development as our future works.

9.5 Running example

9.5.1 ActS

To show the effectiveness of ActS, in this subsection we compare the execution of the rover's mission plan (introduced in Section 9.2) focusing on a specific drive action - `drive(A,B)` - in two different situations: first when ActS is off, so no execution modality is adjusted on-line; second when ActS is on and is in charge of adjusting the execution modality of durative actions while they are in progress. Of course, in both situations the violation of invariant conditions causes the abortion of the current action. Figure 9.7 plots the derivate value of the roll parameter over the time, while the navigate action is in progress in the two situations: when ActS is off (above) and when ActS is on (below). It is apparent that, when ActS is off, the execution of the navigate action is stopped after a number of time instants (in our experiments we sampled the rover's status every second). This happens because of the violation of the invariant conditions associated to the drive, which require that the derivate value of the roll parameter must be below 5 degrees. On the contrary, when ActS is on, the drive action can be carried out successfully; in fact, ActS recognizes a chronicle `hazardous-terrain`, and intervenes at time instant 87 by setting the action modality to *reduced speed*: such a change has a positive effect on the roll derivate, which does not exceed the threshold, and therefore the navigation can go on until the final target (B) is reached. [4]

[4] For the sake of discussion we only consider the roll parameter; however, the interpretation rules used to generate the internal events actually take into account a greater set of parameters.

9.5.2 ActS + Flex-R

Let us consider the case in which ActS is actually combined with the FLEX-R system, and let us imagine what could happen as a consequence of the ActS intervention. For this purpose, suppose that the drive(A,B) is actually just one of a larger set of actions to be executed that actually involves a TP(B) and a COMM(B) as well. The TP(B) is set to perform an high resolution image (HR modality) while the communication has been instantiated to upload data through the CH1. As observed in Chapter 3 (where the planetary rover is introduced) the high detailed image produced by the TP is predicted to consume a larger amount of memory with respect to the low resolution version. While the difference on the time spent by the two versions of the TP is negligible, the particular choice indirectly affects the time spent by the communication action. Indeed as long as the memory to be uploaded increases, the time spent by the communication grows up (actually it is linearly increased).

As defined in Chapter 4, Definition 19, in order for a plan to be valid in our system a plan, it is requested to satisfy some constraint on the use of resources. Let us assume that such requirements constrain the mission to be achieved in less than 107 time slots. Imagine moreover that the mission plan in such a configuration predicts to consume 102 time slots so that the constraints defined for the problem (at least on the time) hold. For the sake of comprehension Figure 9.6 shows the situation before the start of the drive action. As well as there might be many other propositional and numeric fluents, Figure 9.6 focuses on the time and memory resources.

In such a scenario, it is interesting to note that, from the point of view of FLEX-R, the *endogenous event* caused by the intervention of ActS can becomes an issue since the resulting time delay could compromise the consistency of the plan. For instance let us introduce two situations obtained after the execution of the drive action under the ActS control (from Figure 9.6 see *situation A* and *situation B*).

In *Situation A* ActS causes a delay evaluated in 10 time slot; for this reason, the total predicted time moves from 102 to 112. Since the time was constrained to be maximum 107 time slot the mission is not valid anymore in such a configuration. For this reason, after the completion of the drive action, FLEX-R should stop the plan execution to find an alternative configuration of action modalities. A possible solution to this problem concerns the TP to be set to the low resolution modality instead of HR. Imagine that as an effect of this setting the prediction on the time decreases from 112 to 105. Therefore the new allocation is consistent with the numeric constraint and the execution can proceed (indeed $105 < 107$ hold).

In another scenario however the intervention of ActS could have been more prominent (Situation B). If in fact the delay had got a greater impact (122 time

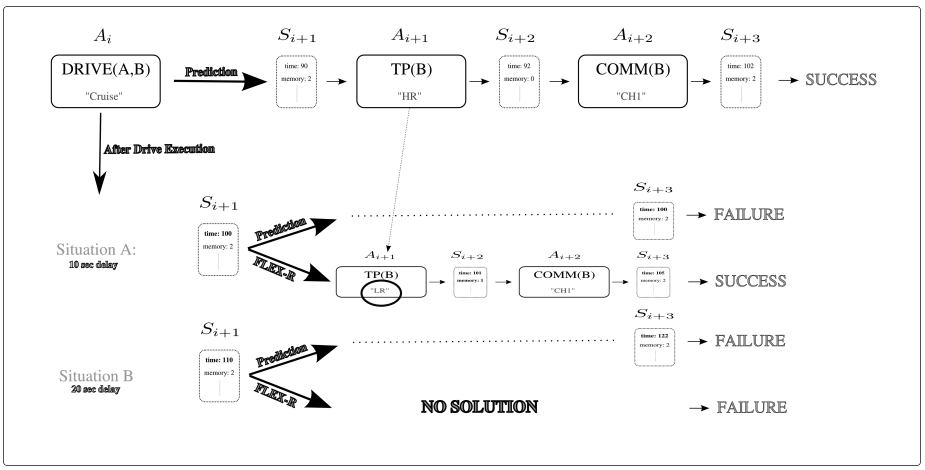


Figure 9.6: ActS + FLEX-R example

slot instead of 102), the FLEX-R would have had no chance for reconfiguring the plan as neither the demotion to the TP toward the low resolution image or the communication set to ch2 would have been able to restore the situation.

It is quite important to note however that given time constraint so strong, even a replanning mechanism could have adjusted the plan. The time indeed is an example of consumable but not renewable resource.

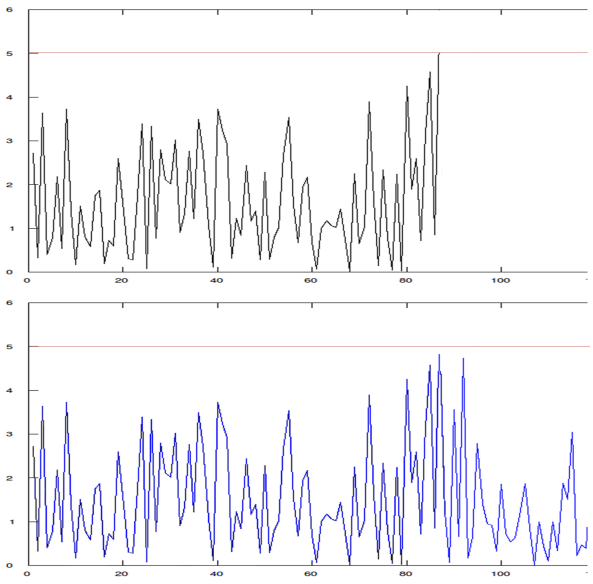


Figure 9.7: The derivate of the roll parameter during action `navigate(A, B)`: when ActS is off (above) and when ActS is on (below).

9.6 Experimental Results

The experimental scenario. The approach described in this Chapter has undergone to a validation by using as test bed the space exploration scenario previously introduced. In particular the role of ActS has been analyzed during the execution of *drive* action. The planetary environment has been modeled as a Digital Elevation Model (DEM); we assumed that an initial DEM D_{init}, presumably computed from satellites images, is available, and we used it for synthesizing a set of rover's missions.

In particular, by taking into account the terrain's characteristics, we have subdivided the rover's missions into two classes: *easy* and *difficult*.

Note that the planning phase verifies the feasibility of each navigate action by invoking a specialized path planner that, relying on D_{init}, for each pair (A,B) of sites of interest, assesses (i) the reachability from A to B, (ii) the distance for reaching B from A, (iii) the validity of the invariant conditions associated with this action type (see Figure 9.1).

Obviously, D_{init} is just an approximation of the real terrain, therefore the actual execution of a mission plan may be affected by unexpected environmental conditions. For simulating the discrepancies between D_{init} and the real terrain, we have altered the original DEM by adding a random noise on the altitude of each cell. In our experiments, we have considered 6 noise degrees: from 10 cm to 15 cm, and for each of them we have generated 320 cases: 160 for the *easy* class and 160 for the *difficult* one.

Note that, since the main contribution of ActS is the supervision of durative actions, we focus the experimental analysis on these actions as they are the most challenging to deal with; in particular, in our experiments we have considered up to 1920 drive actions differing with one another for their starting and ending points, and their length.

To prove the effectiveness of ActS, we have simulated the execution of both *easy* and *difficult* cases in each noisy DEM comparing the responses of the two architectures, a basic one in which ActS is turned on, and an improved one where ActS is switched off. A simplified simulator of the FL has been implemented in order to generate with a frequency of 1Hz the set of raw data the Supervisor (either basic or improved) has to interpret.

For measuring the robustness of the action execution and for providing some insights of the ability of the Supervsior in tolerating uncertainty in the DEM, we are reporting data about three main parameters concerning the execution of the *drive* actions:

1) the percentage of navigate actions that were completed successfully.

2) the percentage of progress actually done by the rover with respect to the whole trajectory, computed taking into account both the navigations that were actually completed and the aborted ones.

3) The percentage of steps the navigation has been performed in the slowdown modality w.r.t. the whole trajectory. Of course this datum is relevant just for the improved architecture.

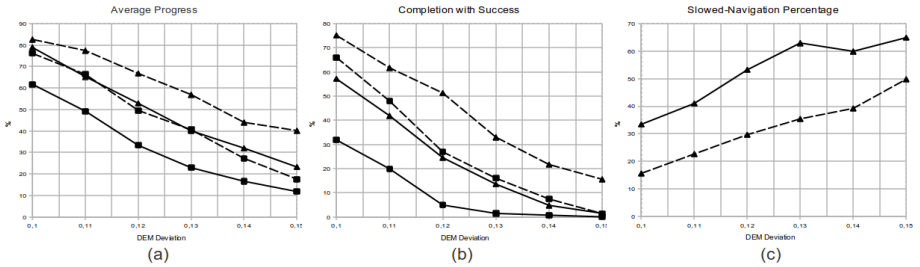


Figure 9.8: ACTS Results

Figure 9.8 summarizes the results of the tests. The graphs show the average values for the class of difficult cases (solid line), and for the class of the easy ones (dashed line). Each bullet corresponds to the average value of 160 navigations; squares denotes the responses of the basic architecture, triangles denotes the responses of the improved architecture.

It is easy to see that the improved architecture always provides better results than the basic one for what concerns both the percentage of success and the progress. Note that, in the difficult cases, the gains are significant even for small deviations in the DEM, whereas in easy ones the gain becomes significant for larger deviations.

The results also show that the mechanism of active control is quite powerful but cannot avoid failures when the noise degree grows.

A final remark concerns the cost of the intelligent monitoring: while the computational cost is negligible, there is an impact on the actual execution that we estimate as the percentage of steps performed in reduced-speed modality, showed in Figure 9.8.c. It is easy to see that this part is small when the noise is small and the cases are easy and increases for difficult cases and larger amounts of error. In conclusion, the overall evaluation shows that the benefit in terms of suceess and progress are quite relevant w.r.t. the part of the path performed at a reduced rate.

9.7 Conclusions

In this Chapter we extend the problem of robust plan execution to deal with durative actions. In particular ActS is able to control the action execution while the action is still in progress.

Similarly to previous works in literature ([5],[76],[19]) where multiple levels of control architecture have been proposed, we have extended the reconfiguration

mechanism provided at plan level by FLEX-RR with a further level of control which focuses on the action level.

As we have seen in the chapter, there is an interesting interdependence between ActS and FLEX-R(R). On one hand, ActS is able to avoid action failures by providing a continuous control over the parameter configuration of the drive action. On the other hand, FLEX-RR is able to compensate the effects of ActS interventions by reconfiguring the whole set of the action whether the ActS contribution (numerically) invalidate the rest of the plan.

Both ActS and FLEX-RR perform action reconfiguration as they rely on the concept of execution modalities, however the flexibility of ActS is limited to the current action configuration while FLEX-RR has the chance of reconfiguring more actions per time. Moreover, to mitigate the ActS intervention we exploit the concept of *Safe Execution Modalities* developed in Chapter 8.

The experimentation phase has been aimed to thoroughly investigate the ActS methodology on a specific navigation scenario within a typical space exploration mission. The results seems promising however further experimentation involving the role of the numeric kernels could be useful. Moreover it would be interesting to employ the mechanism for a new test bed scenario.

However, we believe that the behavior of the actions can be often explained in a temporal dimension by reasoning on the qualitative events happened (e.g. by CRS). To this end, this approach can be useful, as a framework and/or tool, for symbolic control strategy where failures can be prevented by tuning properly and on the fly the action parameter configurations.

Chapter 10

Conclusions

As we have seen in Chapter 2, the technological and methodological advancements in the field of autonomy have provided an in-depth analysis of the problem of robust execution, and a variety of (alternative) approaches have been presented.

In principle the robust execution can be dealt with from the very beginning of the planning process, i.e. in an off-line phase. However, as we have seen, the approach is adequate just when the plan is given in form of a schedule (i.e. as an STP or a DTP, [33],[94],[27],[81]), while for the more general planning context a continual revision of the plan (i.e. on-line) is more suitable and even mandatory when the cases to consider are unbounded. The idea of the continual planning ([34],[15]) involves the interleaving of planning and execution all along the task and, as soon as some plan inconsistency is detected, the attempt to recover from the impasse. Several works based on this idea appear in literature, among which a number of them proposes domain tailored architectures ([5],[96],[76],[19]) while other ones focus on making the plan revision as efficient as possible ([2],[44],[97],[82]).

The architectures that have been developed so far provide good examples for the implementation of real world artificial agents; however the main limit relies on the difficulty of singling out a domain independent methodology. On the other hand, the approaches discussed in the context of repair are quite solid from a methodological point of view, but they are more suitable for the off-line plan-adaptation context. Moreover, even if they proved the efficiency of the repair-based approach against replanning from scratch, they have mainly been employed for the classical setting, and therefore are unable to deal also with consumable resources.

The problem of dealing with resources is receiving an increasing amount of attention in the last years. As a demonstration of this, the planning community has recently introduced the notion of *numeric fluent* ([42]) as a means

for expressing continuous and consumable resources. However, the majority of the methodologies developed so far have addressed the problem of plan generation ([50],[45],[25],[37],[23]). To the best of our knowledge, indeed, previous approaches to robust plan execution that explicitly deal with resources are mainly off-line (e.g., [82, 81, 26]) and only the work presented in [26] actually handles resources that are consumable.

The thesis have addressed the problem of robust execution of plans dealing with continuous and consumable resources, by extending the use of the numeric fluents for the on-line phase.

The main contribution of the thesis is a continual planning framework to (i) detect unexpected contingencies and (ii) - if necessary - recover the plan in execution. The recovery is based on two repair mechanisms: a reconfiguration and a replanning. The continual revision of the plan is the means for achieving tolerance, and hence robustness, in unexpected deviations from the nominal behavior. The methodology have extended hence the classical continual planning framework for dealing with consumable and continuous resources. To achieve this result, the dissertation has addressed the problem from a number of perspectives.

Methodological Contributions. The approach pursued by the thesis handles numeric fluents by introducing the notion of a Multi Modality Action (MMA) (Chapter 3). The model has been conceived observing that in many real world problems an action achieving a given set of qualitative effects can use different configurations of execution, i.e. execution modalities. While these execution modalities have the same propositional aspects, they use and require different amount of resources. For this reason, the MMA models an action by means of two levels of abstraction; the higher expresses the *propositional* high level behavior of the action, while the lower specifies the set of modalities in which the action can be executed.

In other words, the MMA establishes a similarity relation among actions that have the same qualitative effects but can produce and consume resources in different ways. The MMA model is able to capture this similarity in a compact form; even more important such a relation is made explicit inside the MMA model and can be exploited for efficient on-line reasoning mechanism (such as reconfiguration). In this perspective the MMA extends the PDDL model by allowing the user to specify the several execution modalities within a single action schema. Moreover, the MMA is *compatible* with the PDDL model, meaning that the MMA can exploit the several tools available for reasoning about PDDL actions (e.g. automated planner, knowledge engineering tools and so forth).

The MMA proposal fits in the branch of research aimed at extending the PDDL formalisms to handle real world problems (e.g., [49],[38]).

Given the Multi Modality Plan (MMP) characterization, we observed that

while the replanning (i.e. a computation of the MMPP, Chapter 4) is sometime necessary (*invalid plan*), when the plan is just partially valid it may be an excessive reaction. For this reason, we introduced the Dynamic Modality Allocation Problem (DMAP) which allows the agent to manage a *partially valid* plan in a more efficient way.

The DMAP is the declarative representation for the problem of reconfiguring the modality of the actions during the plan execution. To solve the DMAP the thesis have adopted a CSP encoding of the problem and have exploited a general purpose CSP solver. The translation inherits the fundamental steps performed in the CSP based classical planning ([59] and [36]), but extends the mechanism for handling the numeric fluents involved in the MMAs. To the best of our knowledge, the characterization of the repair problem as a process of reconfiguration of action modalities is a (completely) new idea.

Architectures Contributions. Relying on the notion of the MMA and the resolution methods provided by MMPP and a DMAP, the thesis proposes a multi level architecture which integrates in a single framework different mechanisms for dealing with partially knowledge of the environment and unexpected contingencies. The architecture consists of a long term supervisor, namely FLEX-RR, and a reactive module, namely ACTS. The architecture draws one's inspiration from the hybrid architecture proposed by Alami et al. ([5]): indeed it provides in a unified framework deliberation (FLEX-RR) and reaction (ACTS). As an innovation w.r.t. the State of the Art, the architecture is based on a clear domain independent methodology. Indeed, we have adopted the same MMA model throughout the approach, defining precisely the granularity of supervision for FLEX-RR (plan supervision) and ActS (action supervision).

FLEX-RR provides flexibility for the execution of the multi modality plan via the reconfiguration provided by the resolution of the DMAP (in particular by exploiting the CSP mechanism) and a replanning from scratch. Basically, FLEX-RR manages the unexpected contingencies adapting the current course of actions. FLEX-RR contributes with a concrete integration of the reconfiguration and a replanning. Thus, FLEX-RR can take advantage of the (theoretical) more flexibility of a replanner, while relying on the efficiency of the reconfiguration. Moreover, given the high modularity of the system, it can be easily combined with new (possibly more efficient) CSP solver and replanning mechanisms.

ACTS is able to anticipate (some) action failures by reconfiguring the current action execution modality while the action is still in execution. ActS accommodates the execution on the basis of those information which have not been modeled at plan level (for a lack of knowledge or for computational reason) and that represent a threat for the successful completion of the action. The ActS mechanism applies for durative actions and performs the supervision by means of a detection system which classifies and activates the controller module for the

selection of the proper execution modality. While the use of the durative actions has been limited to the off-line phase ([42],[45]), here, ActS have extended the notion for the on-line phase too, by providing a mechanism for the asynchronous control over the action configuration.

Numeric Kernel Contribution. Since the works done in the context of propositional planning, it has been noticed that to avoid plan simulation step, it is possible to annotate the minimal conditions which allow to assess the plan validity ([43], [40]). The result is what is called a kernel. The thesis have generalized the concept for dealing with numeric fluents introducing the notion of Numeric Kernel.

The notion of the kernel is beneficial in FLEX-RR not only because it allows to focus the monitoring only on the numeric fluents which are really relevant, but also because it is possible to implement a focused replanning strategy, namely kernel replanning. This strategy falls into the middle of a spectrum whose extremes are the reconfiguration and the replanning from scratch. The Kernel Replanning turns out to be useful when the resources under consideration are not only consumable but also renewable. A similar approach in literature has been presented in [44], where an anytime repair strategy is introduced. The Garrido's proposal however is just focused on the classical setting and does not account consumable resources.

The kernel has been also useful for a better integration of ActS and FLEX-RR. Indeed, we combined the Numeric Kernel with the MMA model for introducing the notion of a *safe execution modality*. At each step of the execution FLEX-RR can delegate to ActS only the control of those modalities which are actually safe to be executed, meaning that they do not prevent the feasibility of the plan. By this, ActS will prefer safe modalities leaving the other options just for extreme situations.

Experimental Analysis Contributions. To validate the methodology employed, the thesis provides a variety of experimental sessions, which evaluates the FLEX-RR system, the Kernel Replanning and the ActS module (Chapters 7, 8, 9). The domains used as test bed are the *Planetary Rover*, the *ZenoTravel* and the *DriverLog*.

Results confirmed our hypothesis that the reconfiguration is more efficient and still more competent than a replanning mechanism. Even if the replanning is theoretically more competent than a reconfiguration mechanism, in the majority of the tests the replanning was not able to provide an answer given the time limit, while the reconfiguration did. The reconfiguration proved to be feasible also for plan with a large number of actions (40-50 in the Planetary Rover domain) and (60-70 in the *DriverLog* Domain). Moreover, we have seen that the mechanism guarantees a larger plan stability than a replanning from scratch, proving that the strategy could be a viable solution in extending the framework for different

contexts (e.g. mixed initiative system, multi agent systems).

The ActS subsystem has been tested in the *Planetary Rover* domain by checking its ability in carrying on the plan in presence of an increasing amount of discrepancy between the actual and the predicted environment. The experiments have made evident that ActS is able to anticipate many action failures, improving the robustness of the overall plan execution, with a small extra time.

The Kernel Replanning has undergone a first validation in the *ZenoTravel* domain. The result seems rather positive; the kernel replanning outperformed the traditional replanning from scratch methodology in all cases tested. Moreover, surprisingly, the quality of the plan produced by the two mechanisms are quite similar, and even better for the kernel replanning in the hard version of the domain.

A part of the contributions reported in this thesis have been published. In particular in [65] we presented a first prototype of FLEX-RR for the only *Planetary Rover* domain, in [90] we have introduced the notion of the kernel for a repair problem in the *ZenoTravel* domain, and finally in [61] we proposed the ActS subsystem. A preliminary version of ActS has been presented in [64].

Going Further There are a number of extensions that can be investigated starting from the core of the system developed in this thesis. A natural extension of the approach could be the integration in a multi agent system. As highlighted above, one of the advantages of the reconfiguration mechanism is that it keeps the plan quite stable (see Chapter 7). In the thesis we have indeed seen that the DMAP is not allowed to change the structure of the high level plan, since the reconfiguration just changes the action modalities. Typically, in a multi agent system, the main cooperative tasks are modeled at a propositional level. Thus, since the reconfiguration does not change the propositional structure of the (local) plan, whether we employ the mechanism in one of the agents involved, we have the guarantee that its intervention will not endanger the feasibility of future cooperations.

However, real world agents may agree on particular resources usage profiles and a reconfiguration may violate such constraints. For this reason, a multi agent aware reconfiguration should privilege first those modalities which have less impact on the rest of system.

Another possible extension could be the adoption of the reconfiguration for the off-line setting. For instance, the user could be interested in assessing the quality of a given MMA solution by using different criteria of optimization on the available resources, before the actual plan execution. To this end, we have started a preliminary phase of study which has been reported in Chapter 7 where we tested the *DriverLog* and the *Planetary Rover* domain forcing the CSP solver to optimize the overall consumption of resources.

In this thesis we adopted a general purpose CSP engine for the computation

of the DMAP. Despite the performance resulted already quite good, there are a number of possible extensions for enhancing the mechanism. Firstly we could improve the search by adopting a more sophisticated heuristic function tailored on the problem of reconfiguration. Secondly, by exploiting the recent progresses in the reachability analysis performed in the context of off-line planning (e.g. in the FastDownward planning system [48], in the Metric-FF planning system [50]) we could anticipate the CSP resolution by compiling away all the modalities which can be proved to be useless given the problem and the plan at hand.

Analogously to the work presented by Garrido et al. ([44]), it is possible to extend the kernel replanning to consider not only the first kernel at disposal, but all the set of kernels till the goal. However, this kind of mechanism would require some kind of heuristic function to smartly order the kernel to be analyzed. A solution could be the adoption of the distance based heuristic function developed so far in the recent numeric planning system ([50],[45]).

The notion of the Numeric Kernel is rather general in the context of the numeric planning and we believe it can be explored in the more general branch of the case based planning.

As reported in Chapter 4, one of the main advantages of the DMAP relies on the clear distinction between the propositional and numeric aspect of the planning problem. It could be interesting to study an extension of the mechanism where we release this assumption, making explicit the applicability of an execution modality (i.e. its precondition) in the propositional sense too. While in principle the mechanism could be easily handled given the current CSP translation, the efficiency of the system could be compromised because of the occurrence of many interdependences arising between these two levels of representation.

Appendix A

Solving the MMPP: a Top Down Perspective

The more intuitive technique, which can be employed to solve the MMPP, is to invoke a standard PDDL 2.1 planner. As noted previously, we can translate each MMA in a set of flattened PDDL traditional actions (see Figure 3.2). In the PDDL 2.1 formalism this means that the planner has to support the level 2 of PDDL, that is the level in which numeric fluents have been introduced.

Examples of efficient planners in this context are Metric-FF and Lpg-Td that differ in the way the resolution process is performed. While Metric-FF bases its forward search process on the exploitation of a modified version of the relaxation heuristic, the Lpg-Td planning system, instead, searches solutions in the space of action graphs. For further details see [50] and [45].

However, considering the formulation of MMA defined in 3 it may be quite clear that the problem can be considered in a separated fashion. A first phase may in principle search for a solution that is consistent w.r.t. the propositional fragment of the overall MMPP. A second phase can look instead the way in which the plan is actually instantiated in terms of action modalities.

The assumption that the numeric preconditions of a modality do not depend on the propositional high level preconditions allows us to effectively reason in the above terms.

Thus, given a plan of MMA representing a solution of the propositional part of MMPP, we can focus the attention just on the way such actions have to be performed.

More formally we can define the subproblem concerning the modality allocation of the overall MMPP in the following way:

Definition 19. *Given the domain $< U, F, X >$, a Modality Allocation Problem (MAP) Ω is a tuple $< A', \prec, I_{num}, G_{num} >$ where:*

- *A' is a subset of MMA over $< U, F, X >$;*
- *$\prec$ is a total order relation for A'.*
- *$I_{num} \in \Re^{|X|}$ is an assignment of numeric fluents;*
- *G_{num} is the numeric goal; i.e., a conjunction of numeric constraints over X.*

The modality allocation problem corresponds to the consistency checking task for a valid assignment of modalities in A', given the fact that A' is a total ordered set according to $\prec$.

A solution for Ω is an assignment of modalities for each action, i.e. $\{m_0 ... m_{|A'|}\}$ in A' such that the following conditions hold:

- $A_0(m_0)$ is applicable in I_{num} w.r.t. its numeric precondition
- the plan π obtained by considering the total ordered action in A' with the modality $\{m_0 ... m_{|A'|}\}$ is such that $I_{num}[\pi]$ satisfies G_{num}.
- each action $A'_i(m_i)$ is numerically applicable in the state $I[\pi_{0 \to i}]$.

It can be also noticed that a MAP problem is a specific instance of the DMAP problem. In particular, an MAP is equivalent to the DMAP $< \pi, 0, S^0_{num}, G_{num} >$ where of course S_{num} is I_{num}.

Let A' be a solution for the propositional part of the MMPP, it is quite evident that the assignment of A' obtained as MAP resolution, in analogy with the relation among the MMPP and the DMAP, turns out to be a solution for the overall MMPP.

For this reason we have an alternative option for the resolution of the MMPP.

Appendix B

Problem Generators

In Chapter 7 we analyzed the performance of FLEX-RR w.r.t. plans of different length. In particular we were interested in understanding the performance of the DMAP (see Chapter 4). Thus this Appendix explains some detail in the method used to generate cases for stressing the DMAP resolution. In particular we develop two automatic problem generators, one for the *Planetary Rover* and the other one for the *DriverLog* domain.

B.1 Planetary Rover Problem Generation

In the *Planetary Rover*, besides the number of the sites, it is important to select which of these site have to be visited, that is the positions the rover should reach for acquiring information. As such a number increases, the rover is constrained to perform more and more drive actions to reach the spcific locations. Moreover, for each reached site, a take picture has to be performed for actually acquiring the information.

To make the problems more challenging from the numeric point of view, we enforced the rover to transmit all the information acquired at the end of the mission. In addition, we impose constraints both on the overall time and on the total energy consumed.

For very large problems, however, Metric-FF has not been particular efficient (and in some case it has been useless) to generate plans[1]. In particular this happens when propositional and numeric goals show many dependencies among each other. Indeed we was not able to produce plans when constraining the rover to transmit all the information acquired.

For these reasons the plan generation has been performed into two steps. Firstly we gave to the planner a *relaxed* version of the problem in which the only constraint refers to the time and the power spent by the rover. After

[1]We tested Lpg-TD as well noticing the same issues

which we add manually a communication action to the end of the mission, and we imposed a constraint on the memory for explicating that the communication action is actually mandatory. The insertion of the communication action inevitably causes an increasing on the time and on the power, yielding (in the most of the case) a dynamic modality allocation problem.

The tests generated are produced to consider problems ranging among 5 to a maximum of 50 sites. The number of interesting sites (the one that have to be visited) are the half of the total number of sites. They will be randomly selected from the whole set of sites. Distances and the degree of connections among sites have been obtained randomly as well. The power and the time is taken by estimating the average time and power consumption of the rover given the information reported above and the model of the actions.

As anticipated, the number of the sites of the problem is clearly the main parameter for the generation. Since problems have been generated randomly, for each generation, we performed 9 different "runs". For this reason we have a number of 405 ((50-5)*9) pairs "problem and plan".

The DMAP arises when the plan becomes just partially valid. For this reason a part of cases is generated directly at the beginning of the execution because of the insertion of the communication action. Whereas, the other part of cases is obtained simulating the execution of the plan and by injecting discrepancies in the way in which numeric fluents are affected after the action execution.

For the former cases the measure of the difficult of the DMAP corresponds, of course, to the length of the plan. Whereas for the second set of cases, the measure of the DMAP is taken by considering the part of the plan that requires a modalitiy re-allocation. Given the 405 cases mentioned above we collected DMAP ranging from 5 to 50 unities of actions (the number of cities provided a good estimation of the resulting DMAP length).

B.2 DriverLog Problem Generation

Similarly to the *Planetary Rover* domain, the generation of cases for the DriverLog has been performed starting from the number of cities. The other characteristics are generated automatically starting by the number of cities. More precisely:

- the number of packages to be transported is equal to the half of the locations

- each package has to be moved in a location different from the starting position. Both the initial and the final position is randomly chosen on the cities at hand.

- temporal distances (both by walk and by truck) and the degree of connection among the locations is computed randomly
- the total fuel used and overall time consumption are computed by estimating the information above combined to the model of the action.

As the *Planetary Rover* case, the measurement have been performed considering the length of the plans to be reconfigured during the arising DMAP.

The cases have been generated starting by a number of locations ranging from 5 to 30, and for each one of them the generator computed 9 different pairs of problems and plans. The resulting plan to be reconfigured (the DMAP complexity) range between 5 to 80 actions. The total number of cases is 225 ((30-5)*9).

Appendix C

MMA Domains

In this Chapter we report the entire version of the domains used throughout the thesis. The same formulations have been used for the experimental sessions.

C.1 Planetary Rover Domain

Below the PDDL-like code implemented for the representation of the *Planetary Rover* domain.

```
(define (domain planetary-rover)
  (:requirements :typing :fluents)
  (:types robot  - object site - object)
  (:predicates (in ?r - robot ?l - site)
       (road ?l -site ?l1 - site)
       (info ?r - robot ?l - site)
       (infoSent ?r -robot ?l -site)
   )
  (:functions (distance ?l1 ?l2 -site)
      (roughness ?l1 ?l2 -site)
      (comm_cost)
      (time)
      (memory ?r -robot)
      (memoryC ?r -robot)
      (power ?r -robot)
      (powerC ?r -robot)
          (infoLoss)
      (safe_cons ?r -robot)
      (safe_speed ?r -robot)
          (cruise_cons ?r -robot)
```

```
      (cruise_speed ?r -robot)
      (agile_cons ?r -robot)
      (agile_speed ?r -robot)
  (bandwidth-ch1 ?r -robot)
  (bandwidth-ch2 ?r -robot)
  (ch1-cons ?r -robot)
  (ch2-cons ?r -robot)
  )

(:action drive
  :parameters ( ?r - robot ?l1 - site ?l2 - site)
  :modalities (safe,normal,agile)
  :precondition (and (in ?r ?l1) (road ?l1 ?l2)
  (safe: (>= (power ?r) (* (safe_cons ?r)
   (/ (distance ?l1 ?l2) (safe_speed ?r)))))
  (cruise: (>= (power ?r) (* (cruise_cons ?r)
                        (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
  (agile: (>= (power ?r) (* (agile_cons ?r)
                        (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )
  :effect
  (and
    (in ?r ?l2) (not (in ?r ?l1))
    (safe: (decrease  (power ?r) (* (safe_cons ?r)
            (/ (distance ?l1 ?l2) (safe_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (safe_speed ?r)))
           (increase  (powerC ?r) (* (safe_cons ?r)
                           (/ (distance ?l1 ?l2) (safe_speed ?r))))
    (cruise: (decrease  (power ?r) (* (cruise_cons ?r)
                           (/ (distance ?l1 ?l2) (cruise_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (cruise_speed ?r))
           (increase  (powerC ?r) (* (cruise_cons ?r)
                           (/ (distance ?l1 ?l2) (cruise_speed ?r)))))
    (agile: (decrease  (power ?r) (* (agile_cons ?r)
                           (/ (distance ?l1 ?l2) (agile_speed ?r))))
           (increase  (time) (/ (distance ?l1 ?l2)) (agile_speed ?r))
           (increase  (powerC ?r) (* (agile_cons ?r)
                           (/ (distance ?l1 ?l2) (agile_speed ?r)))))
  )

(:action tp
```

```
  :parameters ( ?r - robot ?l1 - site )
 :modalities (lr,hr)
  :precondition (and(in ?r ?l1)
 (lr: (>= (memory ?r) 1))
 (hr: (>= (memory ?r) 2))
  :effect
  (and (info ?r ?l1)
    (lr: (increase (memoryC ?r) 1)
        (decrease (memory ?r) 1)
        (increase (time) 1)
        (increase (infoLoss) 3))
    (hr: (increase (memoryC ?r) 2)
        (decrease (memory ?r) 2)
        (increase (time) 1)
        (increase (infoLoss) 1))))


(:action comm
  :parameters ( ?r - robot ?l1 - site )
 :modalities (ch1,ch2)
  :precondition (and(in ?r ?l1)

 (ch1:   (and (> (memoryC ?r) 0) ( >= (power ?r)
              (/ (memoryC ?r) (bandwith-ch1 ?r)))))
 (ch2:   (and (> (memoryC ?r) 0) ( >= (power ?r)
              (/ (memoryC ?r) (bandwith-ch2 ?r)))))

  :effect
  (and (infoSent ?r ?l1)
     (ch1:   (assign (memoryC ?r) 0)
       (assign (memory ?r) (memoryC ?r))
       (increase (time) (/ (memoryC ?r) (bandwith-ch1 ?r)))
       (increase (powerC ?r) (* (ch1-cons ?r)
       (/ (memoryC ?r) (bandwith-ch1 ?r)))
       (decrease (power ?r)  (* (ch1-cons ?r)
       (/ (memoryC ?r) (bandwith-ch1 ?r)))
       (increase (comm_cost) 1)))
     (ch2:   (assign (memoryC ?r) 0)
       (assign (memory ?r) (memoryC ?r))
       (increase (time) (/ (memoryC ?r) (bandwith-ch2 ?r)))
       (increase (powerC ?r) (* (ch2-cons ?r)
```

```
       (/ (memoryC ?r) (bandwith-ch2 ?r))))
   (decrease (power ?r)  (* (ch2-cons ?r)
   (/ (memoryC ?r) (bandwith-ch2 ?r))))
       (increase (comm_cost) 3))
)
```

C.2 Zeno Travel Domain

```
(define (domain zeno-travel)
(:requirements :typing :fluents)
(:types locatable city - object
aircraft person -locatable)
(:predicates (located ?x - locatable  ?c - city)
             (in ?p - person ?a - aircraft))
(:functions (fuel ?a - aircraft)
            (distance ?c1 - city ?c2 - city)
            (cruise-burn ?a - aircraft)
            (fast-burn ?a - aircraft)
            (inv-zoom-speed ?a - aircraft)
            (inv-cruise-speed ?a - aircraft)
            (capacity ?a - aircraft)
            (total-fuel-used)
       (onboard ?a - aircraft)
            (zoom-limit ?a - aircraft)
            (tot-time)
       (cost)
            )


(:action board
 :parameters (?p - person ?a - aircraft ?c - city)
 :modalities (normal,express)
 :precondition (and (located ?p ?c)
                 (located ?a ?c))
 :effect (and (not (located ?p ?c))
              (in ?p ?a)
(normal: (increase (onboard ?a) 1)
         (increase (tot-time) 240)
         (increase (cost) 1))))
(express:(increase (onboard ?a) 1)
```

```
         (increase (tot-time) 120)
         (increase (cost) 4)))
)

(:action debark
 :parameters (?p - person ?a - aircraft ?c - city)
 :modalities (normal,express)
 :precondition (and (in ?p ?a)
                 (located ?a ?c))
 :effect (and (not (in ?p ?a))
              (located ?p ?c)
(normal: (decrease (onboard ?a) 1)
    (increase (tot-time) 240)
        (increase (cost) 1)))
    (express: (decrease (onboard ?a) 1)
        (increase (tot-time) 120)
        (increase (cost) 1)))
)
(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :modalities (cruise,zoom)
 :precondition (and (located ?a ?c1)
              (cruise: (>= (fuel ?a)
                       (* (distance ?c1 ?c2) (cruise-burn ?a))))
      (zoom: (>= (fuel ?a)
             (* (distance ?c1 ?c2) (zoom-burn ?a))))
 :effect (and (not (located ?a ?c1))
              (located ?a ?c2)
    (cruise:
           (increase (total-fuel-used)
                     (* (distance ?c1 ?c2) (cruise-burn ?a)))
                  (decrease (fuel ?a)
                            (* (distance ?c1 ?c2) (cruise-burn ?a)))
   (increase (tot-time)
             (* (distance ?c1 ?c2) (inv-cruise-speed ?a)))
    (zoom: (increase (total-fuel-used)
                     (* (distance ?c1 ?c2) (zoom-burn ?a)))
                  (decrease (fuel ?a)
                            (* (distance ?c1 ?c2) (zoom-burn ?a)))
   (increase (tot-time)
             (* (distance ?c1 ?c2) (inv-zoom-speed ?a)))
```

```
)

(:action refuel
 :parameters (?a - aircraft ?c - city)
 :precondition (and (> (capacity ?a) (fuel ?a))
                    (located ?a ?c)
)
 :effect (and (assign (fuel ?a) (capacity ?a))
              (increase (tot-time)
 900)))
)
)
```

C.3 DriverLog Domain

```
(define (domain driverlog)
  (:requirements :typing :fluents)
  (:types           location locatable - object
driver truck obj - locatable)

  (:predicates
(pos ?obj - locatable ?loc - location)
(in ?obj1 - obj ?obj - truck)
(driving ?d - driver ?v - truck)
(link ?x ?y - location) (path ?x ?y - location)
(empty ?v - truck))
  (:functions
        (time-to-walk ?l1 ?l2 - location)
        (time-to-drive-fast ?l1 ?l2 - location)
        (time-to-drive-normal ?l1 ?l2 - location)
        (fuel-used)
        (time-spent)
        (fuel-per-minute-fast ?t - truck)
    (fuel-per-minute-normal ?t - truck)
    (load ?t - truck)
        (capacity ?t - truck))


(:action loadtruck
  :modalities (safe,normal)
  :parameters
```

```
   (?obj - obj
    ?truck - truck
    ?loc - location)
  :precondition
   (and (pos ?truck ?loc) (pos ?obj ?loc) (empty ?truck)
        (safe: (< (load ?truck) (capacity ?truck)) )
        (normal: (< (load ?truck) (/ (capacity ?truck) 2))))
  :effect
   (and (not (pos ?obj ?loc)) (in ?obj ?truck) (increase (load ?truck) 1)
        (safe:
             (increase (fuel-per-minute-fast ?truck) (+ (load ?truck) 1))
     (increase (fuel-per-minute-normal ?truck) (+ (load ?truck) 1))
             (increase (time-spent) 4))
    (normal:
             (increase (fuel-per-minute-fast ?truck) (+ (load ?truck) 1))
     (increase (fuel-per-minute-normal ?truck) (+ (load ?truck) 1))
             (increase (time-spent) 2))))

(:action unloadtruck
  :parameters
   (?obj - obj
    ?truck - truck
    ?loc - location)
  :precondition
   (and (pos ?truck ?loc) (in ?obj ?truck) (empty ?truck))
  :effect
   (and (not (in ?obj ?truck)) (pos ?obj ?loc) (decrease (load ?truck) 1)
(decrease (fuel-per-minute-fast ?truck) (load ?truck))
(decrease (fuel-per-minute-normal ?truck) (load ?truck))
        (increase (time-spent) 2)))

(:action boardtruck
  :parameters
   (?driver - driver
    ?truck - truck
    ?loc - location)
  :precondition
   (and (pos ?truck ?loc) (pos ?driver ?loc) (empty ?truck))
  :effect
   (and (not (pos ?driver ?loc)) (driving ?driver ?truck) (not (empty ?truck))
        (increase (time-spent) 1)))
```

```
(:action disembarktruck
  :parameters
   (?driver - driver
    ?truck - truck
    ?loc - location)
  :precondition
   (and (pos ?truck ?loc) (driving ?driver ?truck))
  :effect
   (and (not (driving ?driver ?truck)) (pos ?driver ?loc) (empty ?truck)
        (increase (time-spent) 1)))


(:action drivetruck-fast
  :modalities  (normal,fast)
  :parameters
   (?truck - truck
    ?loc-from - location
    ?loc-to - location
    ?driver - driver)
  :precondition
   (and (pos ?truck ?loc-from)
   (driving ?driver ?truck) (link ?loc-from ?loc-to))
  :effect
   (and (not (pos ?truck ?loc-from)) (pos ?truck ?loc-to)
            (:normal
                   (increase (fuel-used) (* (fuel-per-minute-normal ?truck)
                   (time-to-drive-normal ?loc-from ?loc-to)))
                   (increase (time-spent) (time-to-drive-normal ?loc-from ?loc-to))))
        (fast: (increase (fuel-used) (* (fuel-per-minute-fast ?truck)
                   (time-to-drive-fast ?loc-from ?loc-to)))
                   (increase (time-spent) (time-to-drive-fast ?loc-from ?loc-to))
(:action walk
  :parameters
   (?driver - driver
    ?loc-from - location
    ?loc-to - location)
  :precondition
   (and (pos ?driver ?loc-from) (path ?loc-from ?loc-to))
  :effect
   (and (not (pos ?driver ?loc-from)) (pos ?driver ?loc-to)
```

```
        (increase (time-spent) (time-to-walk ?loc-from ?loc-to)))))
```

Bibliography

[1] Blum A. and Furst M. L. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[2] Gerevini A. and Serina I. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae*, 102(3-4):287–323, 2010.

[3] Agnar Aamodt and Enric Plaza. Case-based reasoning; foundational issues, methodological variations, and system approaches. *AI COMMUNICATIONS*, 7(1):39–59, 1994.

[4] Philip E. Agre and David Chapman. Pengi: an implementation of a theory of activity. In Kenneth D. Forbus and Howard E. Shrobe, editors, *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 268–272. Morgan Kaufmann, 1987.

[5] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, 17:315–337, 1998.

[6] James F. Allen. Planning as temporal reasoning. In *Knoweldge Representation*, pages 3–14, 1991.

[7] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[8] Debdeep Banerjee and Patrik Haslum. Partial-order support-link scheduling. In *ICAPS*, 2011.

[9] Roman Barták, Miguel A. Salido, and Francesca Rossi. New trends in constraint satisfaction, planning, and scheduling: a survey. *Knowledge Eng. Review*, 25(3):249–279, 2010.

[10] Éric Beaudry, Froduald Kabanza, and François Michaud. Planning with concurrency under resources and time uncertainty. In *Proceedings of the*

2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence, pages 217–222, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[11] Sara Bernardini and David E. Smith. Developing domain-independent search control for europa2, 2007.

[12] R. Peter Bonasso, R. James Firby, Erann Gat, David Kortenkamp, David P. Miller, and Marc G. Slack. Experiences with an architecture for intelligent, reactive agents, 1997.

[13] Blai Bonet and Héctor Geffner. Planning with incomplete information as heuristic search in belief space. pages 52–61. AAAI Press, 2000.

[14] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129:5–33, 2001.

[15] Michael Brenner and Bernhard Nebel. Continual planning and acting in dynamic multiagent environments. *Journal of Autonomous Agents and Multiagent Systems*, 19(3):297–331, 2009.

[16] John L. Bresina and Paul H. Morris. Mixed-initiative planning in space mission operations. *AI Magazine*, 28(2):75–88, 2007.

[17] Rodney A. Brooks. How to build complete creatures rather than isolated cognitive simulators. In *Architectures for Intelligence*, pages 225–239. Erlbaum, 1991.

[18] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.

[19] A. Ceballos, S. Bensalem, A. Cesta, L. de Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. van Winnendael. A goal-oriented autonomous controller for space exploration. In *Proc. 11th Symposium on Advanced Space Technologies in Robotics and Automation*, Noordwijk, the Netherlands, April 2011.

[20] Amedeo Cesta, Gabriella Cortellessa, Simone Fratini, and Angelo Oddi. Mrspock - steps in developing an end-to-end space application. *Computational Intelligence*, 27(1):83–102, 2011.

[21] Amedeo Cesta and et al. Profile-based algorithms to solve multiple capacitated metric scheduling problems, 1998.

[22] Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. An iterative sampling procedure for resource constrained project scheduling with time windows. In *In Proceedings of the 16 th Int. Joint Conference on Artificial Intelligence (IJCAI-99*, pages 1022–1029. Morgan Kaufmann, 1999.

[23] Yixin Chen, Benjamin W. Wah, and Chih wei Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *J. of Artificial Intelligence Research*, 26:369, 2006.

[24] Nicolas Chleq. Efficient algorithms for networks of quantitative temporal constraints, 1995.

[25] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*, May 2010.

[26] Amanda Jane Coles. Opportunistic branched plans to maximise utility in the presence of resource uncertainty. In *ECAI*, pages 252–257, 2012.

[27] Patrick R. Conrad and Brian C. Williams. Drake: An efficient executive for temporal plans with choice. *J. Artif. Intell. Res. (JAIR)*, 42:607–659, 2011.

[28] W. Cushing and S. Kambhampati. Replanning: A new perspective. *POS*, page 13, 2005.

[29] Femke de Jonge, Nico Roos, and Cees Witteveen. Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems*, 18(2):267–294, 2009.

[30] Johan de Kleer. Problem solving with the atms. *Artif. Intell.*, 28(2):197–224, 1986.

[31] Lavindra de Silva, Sebastian Sardina, and Lin Padgham. First principles planning in bdi systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09, pages 1105–1112, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[32] Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.

[33] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artif. Intell.*, 49(1-3):61–95, 1991.

[34] Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, and Michael J. Wolverton. A Survey of Research in Distributed, Continual Planning. *AI Magazine*, 20(4), 1999.

[35] Daniel Diaz, Maria D. R-Moreno, Amedeo Cesta, Angelo Oddi, and Riccardo Rasconi. An integrated constraint-based, power aware control system for autonomous rover mission operations. In *Proceedings i-SAIRAS 2012*, 2012.

[36] Minh Binh Do and Subbarao Kambhampati. Solving planning-graph by compiling it into csp. In *AIPS*, pages 82–91, 2000.

[37] Minh Binh Do and Subbarao Kambhampati. Sapa: A multi-objective metric temporal planner. *J. Artif. Intell. Res. (JAIR)*, 20:155–194, 2003.

[38] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trúg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *in Proceedings of ICAPS, 2009*, 2009.

[39] C. Dousson and P. Le Maigat. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI'07*, pages 324–329, 2007.

[40] Richard Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artif. Intell.*, 3(1-3):251–288, 1972.

[41] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[42] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[43] Christian Fritz and Sheila A. McIlraith. Monitoring plan optimality during execution. In *ICAPS*, pages 144–151, 2007.

[44] A. Garrido, Guzman C., and E. Onaindia. Anytime plan-adaptation for continuous planning. In *PlanSIG*, 2010.

[45] Alfonso E. Gerevini, Alessandro Saetti, and Ivan Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artif. Intell.*, 172(8-9):899–944, May 2008.

[46] P. Haslum and H. Ge. Heuristic planning with time and resources, 2001.

[47] Malte Helmert. Decidability and undecidability results for planning with numerical state variables. pages 44–53, 2002.

[48] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[49] Andreas Hertle, Christian Dornhege, Thomas Keller, and Bernhard Nebel. Planning with semantic attachments: An object-oriented view. In Luc De Raedt, Christian Bessiére, Didier Dubois, Patrick Doherty, Paolo Frasconi, Fredrik Heintz, and Peter J. F. Lucas, editors, *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, pages 402–407. IOS Press, 2012.

[50] Jörg Hoffmann. The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. *J. Artif. Intell. Res. (JAIR)*, 20:291–341, 2003.

[51] Jörg Hoffmann and Ronen Brafman. Conformant planning via heuristic forward search: A new approach. *Journal Artifificial Intelligence Research (JAIR)*, 170(6–7):507–541, 2006.

[52] Jörg Hoffmann and Ronen I. Brafman. Contingent planning via heuristic forward search witn implicit belief states. In *ICAPS*, pages 71–80, 2005.

[53] Musso I., Micalizio R., Scala, and et al. Communication scheduling and plans revision for planetary rovers. In *Proc. of i-SAIRAS'10*, 2010.

[54] Meir Kalech. Diagnosing coordination faults in multi-agent systems. *Knowledge Eng. Review*, 24(4):411–412, 2009.

[55] Meir Kalech and Gal A. Kaminka. Coordination diagnostic algorithms for teams of situated agents: Scaling up. *Computational Intelligence*, 27(3):393–421, 2011.

[56] Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997.

[57] Nicholas Kushmerick, Steve Hanks, and Daniel S. Weld. An algorithm for probabilistic planning. *Artif. Intell.*, 76(1-2):239–286, 1995.

[58] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, February 1966.

[59] Adriana Lopez and Fahiem Bacchus. Generalizing graphplan by formulating planning as a csp. In *IJCAI*, pages 954–960, 2003.

[60] Fox M., Gerevini A., Long D., and Serina I. Plan stability: Replanning versus plan repair. In *ICAPS'06*, pages 212–221, 2006.

[61] R. Micalizio, E. Scala, and P. Torasso. Towards robust execution of mission plans for planetary rovers. *Acta Futura*, 5:53–63, 2012.

[62] R. Micalizio and P. Torasso. Plan diagnosis and agent diagnosis in multi-agent systems. volume 4733 of *LNCS*, pages 434–446, 2007.

[63] Roberto Micalizio. Action failure recovery via model-based diagnosis and conformant planning. *Computational Intelligence*, pages no–no, 2012.

[64] Roberto Micalizio, Enrico Scala, and Pietro Torasso. Intelligent supervision for robust plan execution. In *AI*IA*, pages 151–163, 2011.

[65] Roberto Micalizio, Enrico Scala, and Pietro Torasso. Satisfying resource constraints in space missions by on-line task reconfiguration. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2012.

[66] Roberto Micalizio and Pietro Torasso. Monitoring the execution of a multi-agent plan: Dealing with partial observability. In *ECAI*, pages 408–412, 2008.

[67] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *ANNUAL ACM IEEE DESIGN AUTOMATION CONFERENCE*, pages 530–535. ACM, 2001.

[68] Jörg P. Müller and Markus Pischel. The agent architecture inteRRaP: Concept and application. Technical report, German Research Center for Artificial Intelligence, 1993.

[69] Pablo Munoz, Maria D. R-Moreno, Pablo Gallego, and Bonifacio Castano. A cognitive architecture and simulation environment for the ptinto robot. In *Proceedings of the 2011 IEEE Fourth International Conference on Space Mission Challenges for Information Technology*, SMC-IT '11, pages 129–136, Washington, DC, USA, 2011. IEEE Computer Society.

[70] Héctor Muñoz-Avila and Michael T. Cox. Case-based plan adaptation: An analysis and review. *IEEE Intelligent Systems*, 23(4):75–81, 2008.

[71] Nicola Muscettola. Hsts: Integrating planning and scheduling. Technical Report CMU-RI-TR-93-05, Robotics Institute, Pittsburgh, PA, March 1993.

[72] Nicola Muscettola. Computing the envelope for stepwise-constant resource allocations. In *Proceedings of the 9th International Conference on the Principles and Practices of Constraint Programming*, pages 139–154. Springer, 2002.

[73] Karen L. Myers. Cpef: A continuous planning and execution framework. *AI Magazine*, 20(4):63–69, 1999.

[74] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[75] Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artif. Intell.*, 76(1-2):427–454, 1995.

[76] Issa A.D. Nesnas, Reid Simmons, Daniel Gaines, Clayton Kunz, Antonio Diaz-Calderon, Tara Estlin, Richard Madison, John Guineau, Michael McHenry, I-Hsiang Shu, and David Apfelbaum. Claraty: Challenges and steps toward reusable robotic software, international journal of advanced robotic systems.

[77] Tuan A. Nguyen, Minh Do, Alfonso E. Gerevini, Ivan Serina, Biplav Srivastava, and Subbarao Kambhampati. Generating diverse plans to handle unknown and partially known user preferences. *Artificial Intelligence*, 190:1–31, October 2012.

[78] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *J. Artif. Int. Res.*, 35(1):623–675, August 2009.

[79] Y. Pencolé and M.O. Cordier. A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 164:121–170, 2005.

[80] Léon Planken, Mathijs de Weerdt, and Roman van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. In *ICAPS*, 2011.

[81] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen Smith. Solve-and-robustify. *Journal of Scheduling*, 12:299–314, 2009. 10.1007/s10951-008-0091-7.

[82] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. From precedence constraint posting to partial order schedules: A csp approach to robust scheduling. *AI Commun.*, 20(3):163–180, August 2007.

[83] Anand S. Rao and Michael P. Georgeff. Bdi agents: From theory to practice. In *PROCEEDINGS OF THE FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95*, pages 312–319, 1995.

[84] Jussi Rintanen. Complexity of concurrent temporal planning. In *In Proc. of the 17th International Conference on Automated Planning and Scheduling (ICAPS?07*, page 287, 07.

[85] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354, 2004.

[86] Nico Roos and Cees Witteveen. Diagnosis of plan execution and the executing agent. In *Lecture Notes in Artificial Intelligence (LNAI*, pages 357–366, 2005.

[87] Nico Roos and Cees Witteveen. Models and methods for plan diagnosis. *Autonomous Agents and Multi-Agent Systems*, 19(1):30–52, 2009.

[88] Wheeler Ruml, Minh Binh Do, Rong Zhou, and Markus P. J. Fromherz. On-line planning and scheduling: An application to controlling modular printers. *J. Artif. Intell. Res. (JAIR)*, 40:415–468, 2011.

[89] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, second edition, December 2002.

[90] Enrico Scala. Numeric kernel for monitoring and repairing plans involving continuous and consumable resources. In *to appear in ICAART 2013 proceedings*, 2013.

[91] Brennan Sellner, Frederik W. Heger, Laura M. Hiatt, Reid Simmons, and Sanjiv Singh. Coordinated multi-agent teams and sliding autonomy for large-scale assembly. In *Special Issue of the Proceedings of the IEEE on Multi-Robot Systems. In*, page 2006. Press, 2006.

[92] David E. Smith. Choosing objectives in over-subscription planning. In *ICAPS*, pages 393–401, 2004.

[93] Biplav Srivastava, Subbarao Kambhampati, and Binh Minh Do. Planning the project management way: Efficient planning by effective integration of causal and resource reasoning in realplan. *Artificial Intelligence*, 131:73–134, 2000.

[94] Kostas Stergiou and Manolis Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artif. Intell.*, 120(1):81–117, 2000.

[95] Adriaan ter Mors, Jeroen van Belle, and Cees Witteveen. Context-aware multi-stage routing. In *AAMAS (1)*, pages 49–56, 2009.

[96] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. In Martin Buehler, Karl Iagnemma, and Sanjiv Singh, editors, *The 2005 DARPA Grand Challenge*, volume 36 of *Springer Tracts in Advanced Robotics*, pages 1–43. Springer Berlin Heidelberg, 2007.

[97] van der Krogt R. and de Weerdt M. Plan repair as an extension of planning. In *ICAPS'05*, pages 161–170, 2005.

[98] Vandi Verma, Tara Estlin, Ari Jónsson, Corina Pasareanu, Reid Simmons, and Kam Tso. Plan execution interchange language (plexil) for executable plans and command sequences, 2005.

[99] Thierry Vidal and Julien Bidot. Dynamic sequencing of tasks in simple temporal networks with uncertainty. In *In CP 2001 Workshop in Constraints and Uncertainty*, 2001.

[100] Michael J. Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley, 2009.

[101] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. In *In CP*, page 2000, 2000.

[102] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. In *In CP*, page 2000, 2000.

[103] Hakan L. S. Younes and Reid G. Simmons. Vhpop: Versatile heuristic partial order planner. *Journal of Artificial Intelligence Research (JAIR)*, 20:405–430, 2003.