IXTET-EXEC: planning, plan repair and execution control with time and resource management

Solange Lemai

▶ To cite this version:

Solange Lemai. IXTET-EXEC: planning, plan repair and execution control with time and resource management. Automatique / Robotique. Institut National Polytechnique de Toulouse - INPT, 2004. Français. NNT : . tel-00009496

HAL Id: tel-00009496

https://theses.hal.science/tel-00009496

Submitted on 15 Jun 2005

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse

présentée au

Laboratoire d'Analyse et d'Architecture des Systèmes

en vue de l'obtention du

Doctorat de l'*Institut National Polytechnique de Toulouse*
Ecole Doctorale Informatique et Télécommunications
Spécialité : **RCFR**

par **Solange LEMAI-CHENEVIER**

IxTeT-eXeC : planification, réparation de plan et contrôle d'exécution avec gestion du temps et des ressources

---

IxTeT-eXeC: planning, plan repair and execution control with time and resource management

---

Soutenue le 21 juin 2004, devant le Jury composé de :

Raja Chatila	*Président*
Félix Ingrand	*Directeur de thèse*
Malik Ghallab	*Co-directeur de thèse*
François Charpillet	*Rapporteurs*
Maria Fox	
Nicola Muscettola	
Marie-Claire Charmeau	*Invités*
Henri Farreny	
Pierrick Grandjean	

Avant-propos

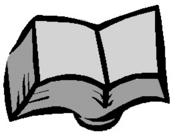
Voilà. Ce manuscrit marque la fin d'une expérience très enrichissante. Maintenant que ce bout de vie s'achève et que je regarde un peu en arrière, j'ai envie de remercier.

> Remercier ceux qui m'ont donné l'occasion de faire cette thèse dans d'aussi bonnes conditions.
>
> Remercier ceux qui m'ont permis de progresser, d'apprendre, d'avoir des idées, d'approfondir, ...
>
> Remercier tous ceux qui ont contribué à faire de ce bout de vie des années sympathiques.

ALORS :

Merci beaucoup à Malik Ghallab et Raja Chatila pour m'avoir permis de m'intégrer et travailler au sein du groupe de Robotique et d'Intelligence Artificielle du LAAS.

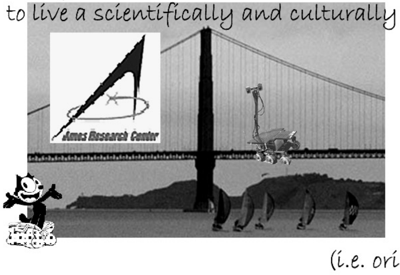
Merci, entre autres, à Philippe David et Luc Planche de EADS-Astrium, à Marie-Claire Charmeau du CNES, pour avoir initié, contribué au financement et suivi cette thèse.



Merci à François Charpillet, Maria Fox et Nicola Muscettola pour avoir pris la peine de lire et commenter mon manuscrit.

Merci à Henri Farreny, Marie-Claire Charmeau et Pierrick Grandjean pour avoir accepté de participer à mon jury.

Many thanks to Nicola Muscettola for giving me the opportunity to live a scientifically and culturally rewarding experience.



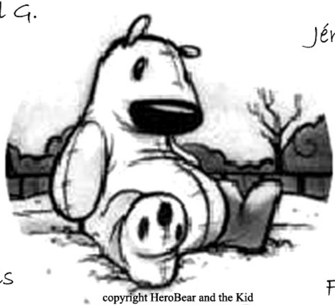
Merci à Félix Ingrand pour m'avoir "dirigée" durant les différentes phases de cette thèse (i.e. orientée, conseillée, corrigée, soutenue, encouragée, rassurée, relue, etc.) avec une disponibilité et une bonne volonté constantes.

Merci, entre autres, à Guillaume Infantes, Anthony Mallet,
Matthieu Herrb, Sara Fleury et Simon Lacroix
pour m'avoir aidée à faire rouler Dala.

Merci à TOUS les membres du groupe RIA
(permanents et générations successives de doctorants et stagiaires)
pour la bonne ambiance qui y règne.

Merci spécial à mes "très chers collègues" (sic) tout simplement pour être devenus des amis :

Romain
Fabien
Emmanuel G.
Jérémi
Williams
Julien
Emmanuel Z.
Juan
Catherine
David
François
Frédéric
Stéphane
Guillaume
[...]



copyright HeroBear and the Kid

Et puis - parce qu'on ne le dit pas assez souvent -
pour être toujours là,
pour pouvoir toujours compter sur eux,
merci :

à mes parents,
à ma famille,
aux "vieux copains",
à ma belle-famille.

D'après "La Promenade" - Marc Chagall

Merci Florent, pour tout.

Sommaire

Contents

Résumé

Les systèmes spatiaux tels que les satellites, sondes et rovers, disposent maintenant de nombreux capteurs, actionneurs et capacités de traitement à bord. Pour des raisons de manque de réactivité, de délais de communication avec le sol, de problèmes de visibilité, il n'est plus souhaitable de les téléopérer entièrement du sol mais plutôt de les contrôler au niveau buts et tâches. Augmenter l'autonomie décisionnelle des sytèmes spatiaux soulève de nouveaux problèmes tels que la planification des activités pour accomplir un but, le contrôle de l'exécution du plan, et la surveillance et le diagnostic de l'état du système.

Nous nous intéressons plus particulièrement à la planification d'une mission et au contrôle de son exécution dans le cadre d'une application avec des contraintes temporelles (rendez-vous avec des fenêtres temporelles de visibilité, etc.) et la gestion de ressources limitées (énergie, carburant, etc.).

Cette thèse propose un cadre général pour combiner la planification délibérative, le contrôle d'exécution d'un plan et l'adaptation réactive d'un plan en exploitant la flexibilité temporelle et le parallélisme des plans produits par un planificateur non linéaire causal (POCL) basé sur des techniques de satisfaction de contraintes, notamment la représentation des contraintes temporelles sous la forme d'un STN (*Simple Temporal Network*). Cette approche a été mise en oeuvre dans le système IxTeT-eXeC qui se compose d'un exécutif temporel et d'un composant de planification qui est basé sur les principes et algorithmes du planificateur IxTeT, modifiés pour prendre en compte la progression du temps et la durée de la planification.

Un premier plan complet est produit puis déroulé par l'exécutif temporel selon un cycle "Perception/Réparation de plan/Action": intégrer les messages externes, réparer le plan si nécessaire, décider des actions à exécuter à l'instant courant.

Deux mécanismes d'adaptation de plan ont été mis en place pour réagir aux événements en cours d'exécution (échec d'une action, dépassement de délai, chûte du niveau d'une ressource, nouveau but). Si le plan est encore temporellement flexible, il est partiellement invalidé en fonction du type d'événement et une *réparation* est tentée pour restaurer les propriétés perdues tout en poursuivant l'exécution en parallèle de la partie valide du plan. Dans d'autres cas, il faut interrompre l'exécution, abandonner le plan en cours et effectuer une *replanification* complète à partir de la situation courante, des buts restants et de leurs contraintes temporelles.

Le système IxTeT-eXeC a été intégré dans le niveau décisionnel de l'architecture LAAS, en interaction avec l'exécutif procédural OpenPRS, pour contrôler un robot mobile autonome avec une mission d'exploration.

Le premier chapitre positionne notre approche par rapport aux systèmes existants. Le deuxième chapitre présente le planificateur IxTeT et les modifications apportées pour améliorer son expressivité et permettre la représentation de domaines plus réalistes (util-

isation de quantités de ressources variables, dépendant de la durée d'une action). Le troisième chapitre détaille le système IxTeT-eXeC: les interactions de l'exécutif temporel avec le système contrôlé, les phases du cycle d'exécution, les conditions formelles pour l'exécution et la réparation simultanées d'un plan et les stratégies d'adaptation de plan. Le quatrième chapitre présente l'intégration dans l'architecture LAAS et les tests effectués sur une plateforme robotique. Le dernier chapitre présente et analyse des tests supplémentaires réalisés en simulation.

Ces travaux ont été réalisés dans le cadre d'une collaboration avec EADS-Astrium et le CNES.

Summary

Space systems such as spacecraft, satellites or rovers embed various and numerous sensors, effectors and processing functions. Due to a lack of reactivity, long communication delays with the ground or restricted visibility windows, these systems can not efficiently be completely teleoperated from the ground. Improving their autonomy requires decisional capabilities such as: on-board planning of the activities necessary to achieve the mission goals, control of the execution of these activities, diagnostic and monitoring of the state of the system.

Our work focused on the processes of planning a mission and controlling its execution in the context of applications requiring the enforcement of temporal constraints (visibility windows, etc.) and the management of limited resources (energy, propellant, etc.).

This thesis proposes a general framework to combine deliberative planning, execution control and reactive adaptation of a plan, that takes advantage of the temporal flexibility and parallelism of the plans produced by a POCL planning system based on CSP managers (particularly, the temporal constraints are handled through an STN). This approach has been implemented in the IxTeT-eXeC system. It is composed of two processes: a temporal executive and a planning system. The latter one relies on the principles and algorithms of the IxTeT planner, modified to take into account the execution context.

A complete plan is produced by IxTeT and then sequenced by the executive according to a "Sense/Plan Repair/Act" cycle: integrate the external messages, repair the plan if necessary, decide when to launch or stop the actions. Two mechanisms of plan adaptation are provided to react to some of the events that may occur during the plan execution (action failure, temporal failure, drop of a resource level, new goal). If there is some temporal flexibility left in the plan, it is partially invalidated according to the type of event and a plan repair process is launched to try to restore the failed properties of the plan, while executing valid actions in parallel. Otherwise, the execution is interrupted and a new plan is produced from the current situation, the remaining goals and their temporal constraints.

IxTeT-eXeC has been integrated in the decisional level of the LAAS architecture, in interaction with the procedural executive OpenPRS, in order to control an autonomous mobile robot with an exploration mission.

The first chapter compares our approach with other existing systems. Chapter 2 describes the underlying IxTeT planning system. We focus especially on its interesting features in a context of plan execution (representation of time and resource, flexibility, plan space search), and present the extension that we made to provide a more realistic and flexible management of resources. Chapter 3 gives a detailed description of the IxTeT-eXeC framework and formally addresses the issue raised by the interleaving of plan repair and execution processes on the same plan. Chapter 4 presents the integration of IxTeT-eXeC in the LAAS architecture and the experiments carried out on a robotic platform.

Finally, chapter 5 presents additional examples and results obtained by simulation.

This work was funded by a contract with EADS-Astrium and CNES.

Chapitre 1

Problématique

Les travaux de cette thèse portent sur les interactions entre la planification d'une mission et le contrôle de l'exécution des plans dans le but d'augmenter l'autonomie décisionnelle de systèmes spatiaux tels que les satellites ou les rovers. Ce type d'application impose de respecter des contraintes temporelles et de gérer des ressources limitées. Dans ce chapitre nous considérons comment ce type de problème a pu être traité. Nous nous concentrons sur deux aspects :

1. Quelles sont les techniques de planification les mieux adaptées dans ce contexte?
2. Comment doivent interagir les processus de planification et d'exécution des plans?

Planifier. . .

. . . pour un domaine lié à une application "réelle"

Prendre en compte le temps

Nous rappelons brièvement les formalismes développés pour représenter et raisonner sur le temps : algèbre des instants, algèbre des intervalles, réseau de contraintes temporelles (TCSP). Nous évoquons l'impact sur la représentation des opérateurs de planification. Deux catégories se distinguent : les approches basées sur une représentation par états et les approches basées sur des chroniques.

Planifier & ordonnancer

Différents types de ressources doivent être pris en compte. Ils se caractérisent par la capacité de la ressource (unaire, multiple) et l'usage qui en est fait (consommation, emprunt, etc.). La gestion de ces ressources implique des mécanismes d'ordonnancement (ordre des actions, allocation des ressources). Dans les planificateurs récents, ces mécanismes sont intégrés dans le processus de planification.

. . . pour une exécution dans un environnement dynamique

Un planificateur pour un système autonome doit prendre en compte une certaine forme d'incertitude qui vient à la fois d'une connaissance approximative de l'état du monde au moment de l'exécution des actions, et des effets non déterministes des actions. Plusieurs techniques de planification ont été proposées pour gérer une partie de cette incertitude :

- les planificateurs “conformant” produisent des plans qui réalisent les buts quelles que soient les circonstances,
- les planificateurs conditionnels produisent des plans contenant des actions de perception et des branches séparées qui seront exécutées en fonction des résultats de chaque perception.

Cependant, l’énumération complète de tous les contextes d’exécution possibles est compliquée par l’introduction d’actions avec durées et effets quantitatifs. Des mécanismes de révision de plans pendant l’exécution demeurent nécessaires. Les plans vérifiant certaines caractéristiques (une flexibilité sur les instants d’exécution des actions et sur leurs paramètres) seront plus robustes à l’exécution.

Notre approche utilise le planificateur IxTeT pour les raisons suivantes :

1. Il permet de gérer le temps et les ressources (représentation à base de chroniques).
2. Il est basé sur une planification non linéaire causale (POCL) combinée avec des techniques de satisfaction de contraintes, produisant ainsi des plans partiellement ordonnées et partiellement instanciés.
3. La recherche dans l’espace des plans partiels peut être adaptée pour effectuer de la réparation de plan.

Intégrer des mécanismes de planification dans un système autonome complexe

Architecture

La conception d’un système autonome repose sur une architecture logicielle. Nous rappelons certaines des architectures classiques qui intègrent des capacités de planification. En résumé, deux niveaux peuvent être distingués : un niveau décisionnel qui génère les plans d’actions accomplissant les buts de la mission et exécute ces plans; et un niveau fonctionnel qui est responsable de l’exécution des actions. Le niveau décisionnel contient deux composants principaux :

- un planificateur qui fournit des mécanismes de délibération pour la gestion de mission à long terme avec des ressources limitées mais qui est souvent coûteux,
- un exécutif plus réactif pour le contrôle de l’exécution des plans.

Dans les sections suivantes, nous développons certaines des stratégies proposées dans la littérature pour faire interagir délibération et réaction.

“Planification réactive” : un exécutif doté de capacités de délibération

L’exécutif est le composant principal. Il contient une librairie de sous-plans précompilés. Le planificateur est utilisé comme une ressource par l’exécutif pour l’aider à faire des choix, pour anticiper en simulant une séquence de sous-plans ou pour générer un nouveau sous-plan correspondant à la situation courante.

“Planification par lots” : planification → exécution → replanification

La planification et l'exécution forment deux processus distincts. A partir de la description d'un état initial, d'un ensemble de buts et d'un modèle, le planificateur génère un plan sur l'horizon de la mission. Ce plan est ensuite passé à l'exécutif qui le séquence et contrôle son exécution. En cas d'échec non réparable par l'exécutif, le planificateur génère un nouveau plan à partir de la situation courante. Le processus de planification est complet mais peut prendre un certain temps, pendant lequel le système reste passif.

“Planification continue” : planification et exécution entrelacées

Les processus de planification et d'exécution sont entrelacés de façon continue. Le processus de planification reste actif pour adapter le plan lorsque de nouveaux buts sont ajoutés ou pour résoudre des conflits après une mise à jour de l'état du système. En outre, les actions qui sont exécutables sont lancées même si le plan n'a pas encore été complètement élaboré.

Planification à tous les niveaux d'abstraction

A l'opposé des approches précédentes qui cantonnent l'utilisation de planificateurs pour l'élaboration de plans d'actions et l'optimisation de l'utilisation des ressources sur un horizon à long terme, un nouvelle approche propose d'utiliser la planification comme mécanisme de raisonnement à tous les niveaux d'abstraction (planification de mission, exécution réactive, . . .).

Finalement, peu d'approches tiennent compte explicitement du temps. Les applications qui nous concernent (actions avec des durées, rendez-vous temporels) nécessitent l'introduction d'un exécutif temporel qui décide des dates de lancement et d'arrêt des actions, surveille la durée des actions et les rendez-vous et réagit a différents types d'échecs temporels. Prendre en compte le temps au moment de l'exécution est délicat (le processus d'exécution lui-même prend du temps), surtout si l'exécution est combinée avec l'adaptation du plan. L'exécutif doit alors contrôler la durée du processus de planification. Une notion de cycle d'exécution apparaît : mettre à jour l'état du système et intégrer les messages, planifier si nécessaire, exécuter les actions à temps.

Notre approche

IxTeT-eXeC est constitué de deux composants qui interagissent sur le même plan : un planificateur et un exécutif temporel. IxTeT-eXeC fournit deux mécanismes pour réagir aux événements survenant pendant l'exécution du plan (échecs ou nouveaux buts) : (1) une réparation du plan en parallèle de l'exécution de ses parties valides, (2) un arrêt de l'exécution du plan et une replanification complète. Ces deux mécanismes sont basés sur la même technique de planification non linéaire (IxTeT), modifiée pour prendre en compte le contexte d'exécution.

IxTeT-eXeC a été intégré dans le niveau décisionnel de l'architecture LAAS, en interaction avec un exécutif procédural.

Le chapitre se termine sur la description d'un exemple de mission pour un rover.

Chapter 1

Context and State of the art

1.1 Problematic

Space systems such as spacecraft, satellites or rovers become more and more complex: they embed various and numerous sensors, effectors and processing functions. Such systems usually execute a sequence of commands elaborated and monitored by experts from a ground station (SOJOURNER), or by mixed initiative planning systems (MAPGEN for MER [Ai-Chang 03]). Due to a lack of reactivity, long communication delays with the ground or restricted visibility windows, these systems cannot efficiently be completely teleoperated from the ground. Improving their autonomy requires to supply decisional capabilities on-board, allowing the ground to teleoperate at a goal level. These decisional capabilities include the choice of activities to perform in order to achieve the goals, the control of the execution of these activities and the diagnostic and monitoring of the state of the system. In 1999, such an autonomous system, called Remote Agent, has been tested in flight by NASA on-board the DeepSpace1 spacecraft [Muscettola 98]. Embarking decisional capabilities is also an active domain of research in "earth" robotics systems such as UAV, Personal Assistant Robots, etc.

Space applications however have stringent requirements. These remote systems have limited resources. For example, the energy is often provided by solar panels and batteries during "shadowed" activities. Thus the system needs to forecast charge and decharge cycles and manage the battery level. Some systems also use propellant (in spacecraft or satellites to perform maneuvers). Such reservoir resource is finite, and the lifetime of the system heavily relies on a careful management of this resource. These systems often embark a payload to make scientific experiments (e.g. cameras). Generally, the memory storage on-board is limited too, and the system needs to download regularly its data to the ground. All these resources and activities need to be properly managed to maximize the mission objectives.

Another main characteristic of space domain is the importance of time. Almost all data are denoted by temporal windows: the visibility windows of ground stations, or, for a satellite, its position on its orbit and the areas it flies over.

We will consider how such a system can be endowed with planning capabilities. The system receives a set of goals from the ground, or establishes new goals "on the fly" (after image analysis for instance) and has a model of its skills: what actions it can perform, what are the conditions to execute them, what are their effects, and how do they interfere. It should decide (or "plan") its course of actions in order to achieve the goals. The system must also be able to execute this plan of actions. A space system is complex and the planning model is often abstract and imperfect. Actions may have non-deterministic effects not accounted for by the model. A space system is also subject to harsh environmental conditions and may evolve in an unknown and dynamic environment (rover). Therefore, it has to monitor the execution of the plan and react to events.

Our work aimed to provide mechanisms to improve the overall robustness of the decisional processes to complete the mission of the system. In the following sections, we discuss how this problem has been tackled so far. We will focus on two main issues:

- What kind of planning techniques is best suited, both in the context of a "real-world" domain, with time and resource requirements, and in the context of plan execution in a dynamic environment?
- How should the planning and execution control processes be linked?

We do not attempt to present an exhaustive list of existing temporal planning techniques and plan execution strategies. Instead, we try to highlight the main difficulties linked to the application requirements and the main characteristics of some approaches, in order to justify the orientation and choices of our own approach.

1.2 Planning...

1.2.1 ...for a real-world domain

Reasoning about time

Classical planning represents actions as instantaneous transitions between states, and goals are not constrained in time. More realistic applications however involve actions with different durations and goals with deadlines.

To introduce temporal reasoning into classical planning, one needs a specific representation of time. Two main formalisms have been developed: the *timepoint algebra* [Vilain 86] and the *interval algebra* [Allen 84]. The first one reasons about instants and qualitative binary constraints between instants (based on the following relation symbols: $\{<, >, =\}$). The second one reasons about intervals and qualitative binary constraints between intervals (using 13 primitive relations[1]). Several subsets of these algebra have been studied and consistency checking algorithms have been proposed.

[1] These primitive relations are: $\{b, m, o, s, d, f\}$ standing respectively for *before*, *meet*, *overlap*, *start*, *during*, *finish*; $\{b', m', o', s', d', f'\}$ standing for *after*, *is-met-by*, *is-overlapped-by*, *is-started-by*, *includes*, *is-finished-by*; and $\{e\}$ standing for *equal*.

The type of applications that we are concerned with also requires to express quantitative numerical constraints. The framework proposed in [Dechter 89] and called *Temporal Constraint Satisfaction Problem* (TCSP) allows the specification of quantitative unary and binary constraints on a set of timepoints: $(a_1 \leq t_i \leq b_1) \vee \ldots \vee (a_n \leq t_i \leq b_n)$ and $(a_1 \leq t_j - t_i \leq b_1) \vee \ldots \vee (a_n \leq t_j - t_i \leq b_n)$ with $a_1, b_1, \ldots, a_n, b_n \in \Re$. The general problem takes into account disjunctions of constraints on the distance between two timepoints. A simplified version of the problem, called *Simple Temporal Problem* and represented as a *Simple Temporal Network* (STN) is commonly used by temporal planners to reason about temporal variables and constraints [Laborie 95b, Rabideau 99, Frank 03]. The nodes in an STN represent timepoints and the arcs represent duration constraints between two timepoints. Arcs are labeled with a single numeric interval. Efficient algorithms exist to add constraints and check the network consistency.

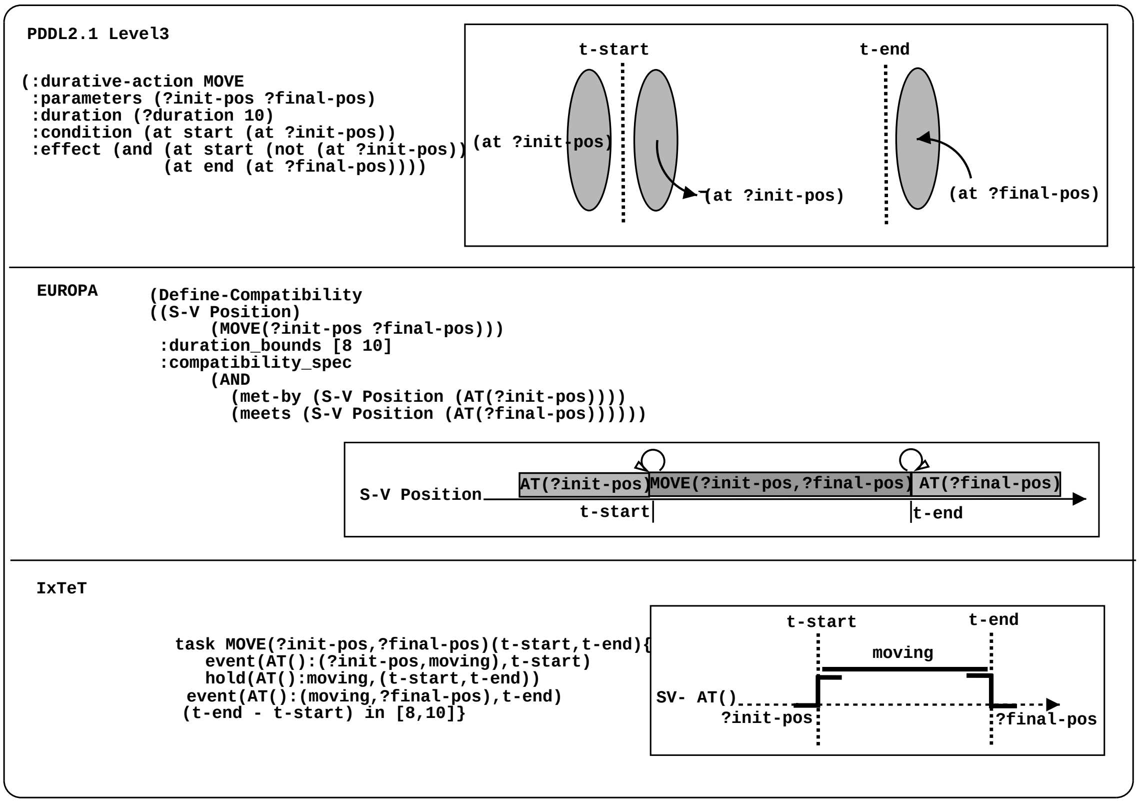
Furthermore, the handling of time impacts the representation of planning operators. In a STRIPS formulation [Fikes 71], actions and their associated preconditions and effects are instantaneous. An extended version (e.g. used in the DEVISER planner [Vere 83]) assumes that preconditions have to be true at the start and remain verified throughout the action, whereas effects are undefined during the action and become true at the end. The PDDL2.1 Level3 language [Fox 02b], used during the AIPS'02 planning competition [Fox 02a], improves the expressiveness by allowing the specification of local preconditions and effects on start and end timepoints of a durative action, as well as invariant conditions to maintain throughout the action. PDDL2.1 also supplies the use of numeric variables which value can be accessed and updated instantaneously. These improvements allow the expression of resource usage and a weaker mutex relation[2].

These languages share a common representation of an action as a transition between two global states. Intermediate timepoints inside the action cannot be specified, and joint effects of actions cannot be addressed. This type of representation has been widely used with various planning techniques. Examples include TLplan [Bacchus 01] and TALplan [Doherty 01] in state-space planning, TGP [Smith 99] and SAPA [Do 01] in planning graph approaches, O-Plan[Currie 91] and SHOP2 [Nau 01] in HTN planning and [Coddington 02] in plan space planning.

In another approach, called "time-oriented" as opposed to the previous "state-oriented" one, the world is described as a set of state variables which are functions of time. A timeline, associated with each state variable, describes the history of changes and persistences of the state variable's value over the planning horizon. A plan is made of the set of timelines. Several planners are based on this time-oriented approach. We will introduce two variants:

- A timeline in the Europa planner [Frank 03] and its predecessor HSTS [Muscettola 94] consists in a sequence of *token* intervals representing either a state (persistence) or an activity (change of state). Domain constraints define for each token the set of

[2] Two actions can be concurrent if there is no interactions between the timepoints of an action and the timepoints (if simultaneous) or the invariants of the other action.



PDDL2.1 Level3

```
(:durative-action MOVE
 :parameters (?init-pos ?final-pos)
 :duration (?duration 10)
 :condition (at start (at ?init-pos))
 :effect (and (at start (not (at ?init-pos)))
              (at end (at ?final-pos))))
```

EUROPA

```
(Define-Compatibility
((S-V Position)
      (MOVE(?init-pos ?final-pos)))
 :duration_bounds [8 10]
 :compatibility_spec
      (AND
        (met-by (S-V Position (AT(?init-pos))))
        (meets (S-V Position (AT(?final-pos))))))
```

IxTeT

```
task MOVE(?init-pos,?final-pos)(t-start,t-end){
   event(AT():(?init-pos,moving),t-start)
   hold(AT():moving,(t-start,t-end))
  event(AT():(moving,?final-pos),t-end)
 (t-end - t-start) in [8,10]}
```

Figure 1.1: *Example of various representations for a durative action*

configurations in which its insertion in the plan is valid.

- A timeline in IxTeT [Laborie 95b] is a sequence of temporal assertions that can represent either an instantaneous change of values or the persistence of a value over an interval. In the IxTeT formalism, an action describes the partial evolution of a subset of state variables over the action's duration interval.

Figure 1.1 illustrates and compares the description of an action in the state-oriented approach and in the two variants of the time-oriented approach. We assume in this example that the action move(?init-pos,?final-pos) only affects the position of the robot (the other characteristics of the world remain unchanged). Note that the action duration in this example is expressed as a numerical value or interval. However, in the three cases, more elaborate relations can be expressed between the duration and the effects of the action (e.g. the duration can be computed according to the distance between the two locations and the speed of the robot). The time-oriented representation allows the expression of intermediate timepoints and more complex synchronization mechanisms between actions. The time-oriented approach and its use in temporal planning are thoroughly discussed in [Ghallab 04].

Planning & Scheduling

Achievement of goals requires the execution of actions ("what" needs to be done), the selection of agent and resources that execute them ("how" the actions should be done) and the determination of time of occurrence and action ordering ("when" the actions should be done). Classically, **planning** deals with the determination of "what" to do and "when"according to causal constraints, whereas **scheduling** refers to the determination of "when" to schedule precisely the actions and "how" to allocate the resources. Autonomous spacecraft or robot applications involve both planning and scheduling problems: the system may choose actions in order to achieve a new goal, while assigning limited resources to actions.

Different types of resource and resource usage need to be considered. **Reusable resources** are "borrowed" by an action and released when it finishes. Such a resource can have a multiple capacity (e.g. power) and concurrent actions requiring this resource are allowed providing that their cumulative usage does not exceed the resource capacity. Moreover, a unary reusable resource, such as a camera, imposes a total order between actions requiring it. **Consumable resources** can be consumed and possibly produced by an action. A reservoir resource (e.g. fuel tank) is a consumable resource which has a maximum capacity and may have an initial level. Planning with actions requiring reservoir resources imposes to check for each resource that the level profile over the plan horizon never exceeds the capacity or becomes negative.

In [Smith 00], the authors describe several scheduling and classical planning techniques and discuss how these planning techniques have been (not so easily) extended to handle time and/or metric resources. Recent planners take into account the action scheduling and resource allocation problems while planning. Some planners embed Operation Research techniques such as linear programming (Zeno [Penberthy 94]) or integer programming [Vossen 99] to manage metric quantities.

A promising framework to handle such problems seems to be the use of a time-oriented approach combined with the use of underlying constraint networks and constraint satisfaction techniques (an STN for instance allows continuous time handling). Examples of such planners include HSTS, Europa and IxTeT. Useful algorithms for propagating resource constraints in a CSP-based, integrated planning and scheduling framework can be found in [Laborie 01].

1.2.2 ...for execution in a dynamic environment

An autonomous agent such as a robot or a spacecraft is a complex system and evolves in a dynamic and unpredictable environment. The planning process of choosing which actions to do in order to achieve a goal has to deal with uncertainty. This uncertainty comes both from an incomplete and incorrect knowledge of the state at execution time, and from the non-deterministic effects of the actions. Several planning methods have been developed to handle part of this indeterminacy.

Conformant planners (CGP [Smith 98], CMBP [Cimatti 00], C-PLAN [Ferraris 00]) produce plans that achieve the goals in all possible circumstances. Such a plan contains only actions that are applicable and lead to the goal whatever the current state is. Consider for instance a "painter" robot. Initially, there are two objects (O_1, O_2) and several cans of paint, with everything of unknown color. The goal is to have both objects painted in the same color. A conformant plan would consist in picking up one can and painting both objects using this can. Being quite restrictive, conformant planning is often practically inapplicable.

Conditional (or contingent) planners (SGP [Weld 98], MBP [Bertoli 01], GPT [Bonet 00], Shaper [Guéré 01]) produce plans with sensing actions. After each sensing action, the plan contains separate branches corresponding to the different plan continuations, according to the different possible results of the state observation. The painter robot for instance will first sense the color of both objects. If they are the same color, the goal is achieved. If not, the robot senses the colors of the cans. If one can has the same color as one object, then it paints the other object with that can. If not, it chooses one can and paints both objects. Contingent planning can be computationally too expensive to be performed on-board a spacecraft or robot within the limited computational resources (memory, processor speed and time to react) typically available to control these systems.

In a more complex domain involving for instance durative actions and quantitative effects (resource usage), the set of all possible situations is too large to be enumerated. In such a context, the agent needs to monitor the execution and revise the plan when necessary. Thus, in [Washington 99], a contingent plan is uplinked to a planetary rover, as well as alternate plans used during execution to perform failure recovery. We will introduce in section 1.3 other existing strategies for plan adaptation during execution.

Nevertheless a certain amount of plan revisions can be avoided depending on the planning techniques used to generate plans. Plans with the following properties are especially worthwhile in an execution context:

- **order-constrained plans** – Some metric temporal planners ([Do 01], [Bacchus 01], [Smith 99], [Edelkamp 01]) produce *position-constrained* plans which specify the exact start time for each action. Some others ([Penberthy 94], [Muscettola 94], [Laborie 95b]) produce *order-constrained* plans which only specify precedence constraints between actions. Such plans provide a better execution flexibility and an improved makespan. Indeed, an order-constrained plan corresponds to a set of position-constrained plans, each being suitable for execution. In [Do 03], the authors propose a post-processing method to generate an order-constrained plan from a position-constrained temporal plan.
- **plans with unbound variables** – The parameters of the actions remain flexible and the plan only specifies separation constraints between variables. Similar to the temporal flexibility in order-constrained plans, the parameters of the actions can be

further constrained during execution. If the painter robot has two brushes, and one object to paint, the action paint(O_1,$?b$) with brush $?b$ in $\{B_1,B_2\}$ remains executable even if one brush mysteriously disappears. If the robot has two objects to paint, an order-constrained plan with unbound variables contains two actions paint(O_1,$?b_1$) and paint(O_2,$?b_2$), linked by the constraint $?b_1 \neq ?b_2$, that can be concurrently executed.

1.2.3 Conclusion

We opted for the planning/scheduling system IXTET for the following reasons:

1. It represents and reasons about continuous time and metric resources. It is based on a "time-oriented" approach, with a description of the domain through state variables. Ordering constraints, numeric temporal constraints and resource metric constraints are managed using constraint networks and constraint satisfaction techniques.
2. It is based on a Partial Order Causal Link (POCL) planning process combined with CSP-techniques, thus producing order-constrained plans with unbound variables and providing execution flexibility.
3. Search in the partial plan space can be adapted to perform plan repair. Since plans possibly contain parallel actions that are executed concurrently, interleaved plan repair and execution can be envisioned.

Chapter 2 describes in more details the IXTET planning system and the modifications we made to make it more suitable for plan execution.

1.3 Embedding planning in a complex autonomous system

1.3.1 Architecture

The definition of a software architecture contributes towards designing, developing and deploying complex autonomous systems. An architecture describes the different components (from basic built-in action and perception functions to reasoning capabilities) and how they interact to perform the task of the system. Among the various approaches proposed for autonomous mobile robots and spacecraft, we will focus on architectures embedding planning capacities.

First approaches were based on a simple "Sense/Plan/Act" open-loop: sense the world, translate the data into a world model, generate a plan to achieve a goal based on this model and execute the plan (cf. Figure 1.2(a)). Shakey [Fikes 72] was the first robot to integrate perception, planning and execution. The PLANEX system on-board Shakey used the STRIPS planner to generate plans and triangle tables to monitor the execution. These tables allowed basic failure recovery: if an action fails and its preconditions are still valid,

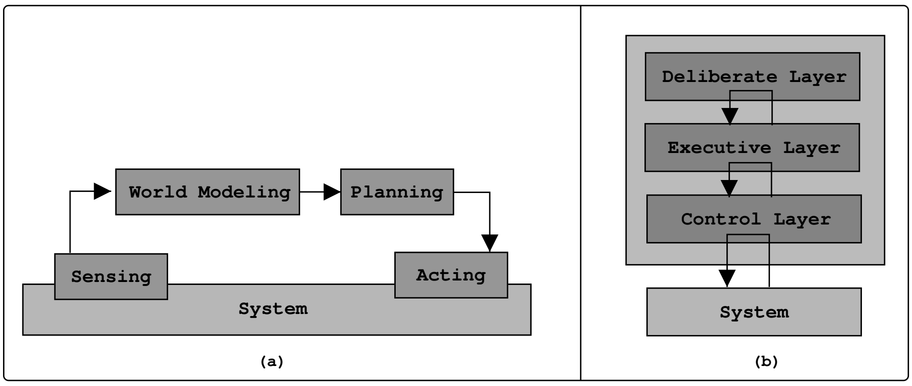


Figure 1.2: (a) *The "Sense/Plan/Act" open-loop,* (b) *The three-layered architecture*

the action is launched again, otherwise a new plan is generated. This open-loop plan execution proved to be inadequate in dynamic environments.

More recent works rely on an approach, called P-SA in [Kortenkamp 98], in which the system *"plans based on initial conditions and common knowledge (P) and then executes this plan using sense-act (SA) behaviors, replanning only when the reactive behaviors run out of routine solutions"*.

Many of these architectures rely on three layers. E. Gat highlights in [Kortenkamp 98] the equivalence between the 3T ([Bonasso 97]) and ATLANTIS ([Gat 92]) architectures: the three layers consist of three components which are a reactive feedback control system, a reactive plan execution system and a time-consuming deliberative system (cf. Figure 1.2(b)).

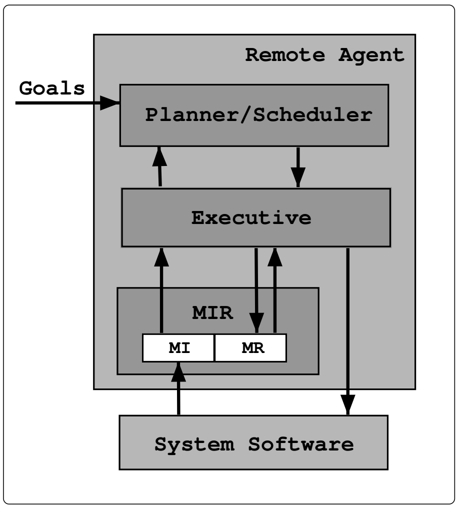


Figure 1.3: *Remote Agent architecture*

The architecture deployed for the Remote Agent experiment [Muscettola 98] on-board the DeepSpace1 probe in 1999 could be described as a three-layered architecture with three main components (cf. Figure 1.3). A planner/scheduler based on the HSTS planning system [Muscettola 94] is responsible for back-to-back mission planning and replanning. An executive based on ESL [Gat 97] refines the actions in the plan and eventually tries alternative methods to recover from a failure. Finally a diagnostic system, called MIR (*Mode Identification and Recovery*) and based on the Livingstone technology [Williams 96] estimates the state of the system from the sensor data and can suggest recovery actions to the executive.

The LAAS architecture [Alami 98] is also decomposed into three layers, with several components: a functional level contains a set of modules encoding the basic functionalities, a decisional level is responsible for both plan generation and plan execution, and the in-between execution control level has a fault protection role and filters the commands sent to the functional level. Our temporal planning/execution system IxTeT-eXeC has been integrated in the decisional level. This integration is detailed in chapter 4.

A "two-layer" architecture is being developed at Jet Propulsion Laboratory, called CLARAty (*Coupled Layer Architecture for Robotic Autonomy*) (see [Volpe 00]). The functional layer embeds communicating objects encoding basic functionalities. Each object has the possibility to predict its resource consumption and provide this information to the decisional level as requested. The decisional level contains two interacting components: a planner/scheduler (based on a declarative model) and an executive (based on a procedural model). In the long-term, the authors plan to have a common plan database based on a representation language combining procedural and declarative structures. Meanwhile, the CASPER [Chien 00] and CLEaR [Estlin 01] systems are presented as instances of this decisional layer. CASPER associates the planning system ASPEN [Rabideau 99] with a simple executive and CLEaR associates CASPER with an executive based on TDL [Simmons 98].

As seen in the previous examples, the number, role and components of layers in an architecture vary. Generally speaking, we can consider two main levels. A **decisional level** is in charge of generating a plan of actions to achieve mission goals, and executing the plan in a robust way. A **functional level** is in charge of executing the actions of the plan. It is interfaced with the hardware. The execution of an action may require diverse processing functions and control loops. This level also interprets and merges sensor data to provide the above level with the system state. Indeed the decisional level needs a knowledge of the global state of the system, whereas functional processes reason on subsets of this state.

Managing long-term missions with limited resources requires deliberation mechanisms which are often computationally expensive, whereas evolving in a dynamic environment requires a reactive behavior. Thus the decisional level is broken down into two main components: a **planner** and an **executive**. The planner, based on a declarative model, the system state and a set of goals, produces plans which execution should achieve the goals. The executive is in charge of executing and monitoring plans (predefined and/or automatically generated). It has a time-bounded reaction to events (such as using precompiled error handlers in response to execution failures).

We are interested in the key problem of defining how the two components should interact to balance deliberation and reaction. Various strategies have been applied so far. In section 1.3.2 we discuss approaches based on a strong executive and little planning. Section 1.3.3 presents an example of batch-oriented planning and its limitations. In section 1.3.4 we discuss some of the mechanisms proposed to interleave planning and execution in a continuous way. Finally section 1.3.5 describes recent and innovative work on the use of temporal planning techniques at various levels of the architecture.

1.3.2 Executive enhanced with deliberative capabilities or "reactive planning"

The executive is the main component and is given a set of pre-compiled subplans. The planner serves as a resource for the executive to help it make choice, to anticipate by simulating a sequence of subplans or to generate a new subplan corresponding to the current situation.

Examples of "simple" executives include RAP, ESL and TDL. The *Reactive Action Package* system [Firby 94] has been designed for the reactive execution of the tasks of a symbolic plan. For each task, the RAP system selects a refinement method in a set of predefined methods according to the situation. Concurrent task execution is allowed and synchronized with events (reports from the functional level). In case of failure, a new method is selected, and so on, until the task is completed.

The *Execution Support Language* [Gat 97] provides constructs to encode execution knowledge in autonomous agents, including features such as: contingency handling, task management, goal achievement and logical database management. Contingency handling in ESL is similar to RAP. Multiple outcomes of an action are categorized as success or failure. The system responds immediately to failures by applying recovery procedures.

The *Task Description Language* [Simmons 98] also provides support for task decomposition, synchronization, execution monitoring and exception handling. A task is executed following a task tree generated dynamically according to the current perception data. Exception handlers are associated with nodes in the tree and specific "reasons". In case of failure, an exception handler that matches the failure reason is looked for and invoked. It can eventually fail in recovering the error, and the search for a matching exception handler continues up the tree.

These systems fail if they face a situation for which no explicit method or procedure is provided. The following approaches propose to extend an executive with deliberative capabilities.

In PROPEL [Levinson 95], the executive relies on predefined programs for routine situations, and uses predictive search to evaluate choices and generate procedures in unexpected situations. Propel embeds two main components: an executive and a planner. Both use the same procedural search engine. Being reactive, the executive does not perform backtracking through the search tree, whereas the planner can backtrack on previous choice points. This planning ability is used with two aims: (1) anticipation and (2) unexpected failure handling. (1) The planner is extended to simulate sensing and effecting and produce a plan corresponding to a set of choice selection rules associated with a top-level procedure. These rules guide the executive in the choice points encountered during the procedure execution. (2) When the executive is frozen by an unexpected failure, the planner performs backtrack through the search tree originally generated by the executive and provides new selection rules, used by the executive to handle the situation.

Similar functions have been implemented in Propice-Plan [Despouys 99]. A procedural executive (OpenPRS[3]) is endowed with plan synthesis and anticipation planning. The executive requests plan synthesis when encountering a new goal for which no procedure is available. Planning amounts to selecting and combining existing procedures to achieve the goal from the current situation. During the executive's idle time, future choices are simulated. This anticipation process advises the executive in the selection of the best option and forecasts problems.

Another approach worth mentioning is the use of *transformational planning* by an executive. XFRM [Beetz 94] for instance has been developed and tested on indoor robot applications. The robot's executive is provided with a library of plans specified in the procedural language RPL. There is at least one default plan for each goal sent to the executive. XFRM also has a taxonomy of failure models. The default procedure is executed if the executive has no time left to reason. Otherwise, transformational planning diagnoses "bugs" in the default plan (future possible plan failures) and tries to improve it. Planning is thus a search in the space of complete and executable plans. A search step is divided into three phases: projection of execution scenarios, diagnosis of projected plan failures and plan revision via a set of transformational rules. These revisions are not guaranteed to improve the plan or even eliminate the bug, but potentially make it more robust.

Still, the systems presented above do not perform a projection of the state far in the future. This look-ahead is yet necessary to manage the level of a limited resource (such as power or memory space) during the entire mission. It is then desirable to have a long-term plan that predicts the evolution of the resource over the mission horizon.

In [Washington 99], the authors propose an architecture for a planetary rover without on-board planner. Instead, a library of plans (which actions consist in low-level commands) is uploaded to the on-board executive. This library contains a long-term plan with contingent branches and a set of alternate plans. The executive sequences the plan and selects a branch in the plan according to a utility function. During execution, a resource manager regularly compares a resource actual usage and predicted usage to the resource availability profile and detects current conflicts or potential future conflicts. Similarly, a model-based estimator regularly updates the state knowledge of the executive. In case of failure and in addition to the contingent branches, the executive can use alternate plans to adapt locally the plan and recover, or to switch to an overall backup plan. If no alternate plan is applicable, the planner on the ground is still needed.

Moreover, procedural executive such as TDL, ESL or PRS [Georgeff 89] do not handle quantitative temporal constraints and scheduling problems. A new approach, presented in [Kim 01], proposes a language RMPL (*Reactive Model-based Programming Language*) along with a mixed planning/execution system called Kirk. RMPL aims at combining

[3] This executive is integrated in the LAAS architecture and frequently used on-board indoor and outdoor robots. It is described in chapter 4.

in a unified representation the flexibility of reactive execution languages and the forward looking scheduling ability of temporal planners. RMPL programs are compiled by Kirk into *Temporal Plan Networks* (TPNs). A TPN is a *Simple Temporal Network* [Dechter 89] augmented with symbolic constraints and decision nodes. The arcs are labeled with the classical temporal constraints and new symbolic constraints: condition assertions or requirements. Specific decision nodes encode multiple strategies and contingencies (choices are made by the planner). The online planning algorithm searches for unconditional, temporally consistent paths in the TPN, thus producing an order-constrained temporal plan. This plan is then executed following the plan runner strategy used on-board the Remote Agent [Tsamardinos 98]. This approach does not yet provide resource management, neither replanning or exception handling although RPML allows the expression of exception handlers. This extension would require more expensive conditional planning algorithms.

Finally, these "smart" executives are useful to provide reactive behaviors but an on-board planning process may still be required to handle unpredicted events, or to manage resource constraints and temporal constraints (such as deadlines) over the mission horizon.

1.3.3 Planning → execution → replanning or "batch planning"

In this framework, planning and plan execution are two separate and successive processes. Given an initial state, a set of goals and a model, the planner generates a plan ranging over the mission horizon. This plan is then passed to the executive which sequences it and monitors its execution. If a failure occurs and cannot be recovered by the executive, the planner is requested to replan, i.e. generate a new plan from the current situation and the updated set of goals. The *replanning* process is generally complete (if a solution exists, it will find it) but may take a lot of time. And in the meantime, the system has to stay in a standby mode.

The Remote Agent [Muscettola 98] was also performing back to back planning. Time is broken into a set of planning horizons. The current plan contains a planning action near the end of its horizon. The executive achieves this action by transmitting the request to the planner, along with an estimation of the resource constraints and of the initial state of the following plan (corresponding to the future state at the end of the current plan). However, it is difficult to anticipate the maximum time needed by most planning systems to come up with a plan, and hence to foresee the duration of this planning action. The "future initial state" may also deviate from the predicted one (therefore, this state is required to correspond to a standby one).

1.3.4 Interleaved planning and execution or "continuous planning"

Continuous planning implies that planning and plan execution processes are seamlessly interleaved. The planning process remains active to adapt the plan when new goals are added or to resolve conflicts appearing after a state update. Besides, actions that are ready to be executed are committed to execution even if the plan is not yet completely generated.

Planning, execution and monitoring have even been integrated in the same process in the IPEM framework (*Integrated Planning Execution and Monitoring*) [Ambros-Ingerson 88]. IPEM is based on a hierarchical partial-order planning technique. Using this technique, a solution plan is found by fixing all flaws in a partial plan. Flaws are classically: unsupported preconditions of actions, conflicts between actions or unexpanded actions. Each flaw is associated with a specific fix or plan transformation. IPEM extends this set of {IF <flaw> THEN <fix>} rules to handle execution and monitoring:

- Execution is pursued thanks to two new rules: (1) {Unexecuted Action/Execute} and (2) {Timed-Out Action/Excise action}. (1) An action is executed if its preconditions are supported and if it is not in conflict with a "live" action (the plan is order-constrained thus allowing concurrent execution). (2) The action is completed and removed from the plan.
- Monitoring is done by detecting redundant actions (their effects are already present in the current state description) and unsupported protections. The plan is then fixed by removing actions or asserting new protections.

The search in the partial plan space is guided by a flaw ordering heuristic. It is important to note that the use of these new "execution flaws" leads to non-backtrackable executed fixes, thus compromising the completeness of the search process.

The SPEEDY system [Bastié 96] presents similar features. The main differences rely in the use of a linear planning technique to help in determining the executable actions, coupled with a triangle table used to eventually execute actions in parallel.

The ROGUE system [Haigh 98] makes up the decisional level on board Xavier the robot. It is composed of a task planner (PRODIGY4.0), an execution processing module that monitors the environment for exogenous events, and a learning module. PRODIGY [Stone 96] is a domain-independent nonlinear state-space planner that uses means-ends analysis and backward chaining to reason about multiple goals and multiple alternative operators to achieve the goals. It has been extended to handle asynchronous goal requests and interleave planning and plan execution. Replanning proceeds if there exist not satisfied top-level goals or if execution monitoring reveals a failure situation. Each time the planner generates an executable plan step, the action is executed and monitored. In case of failure, the state knowledge is updated and the next plan step is generated. An interesting feature of the ROGUE system is its ability to learn from real execution to improve planning. Robot's execution traces are analyzed to identify situations in which the planner's behavior needs to change. Situation-dependent rules are created and used by the planner to select between alternatives.

SIPE (*System for Interactive Planning and Execution monitoring*) [Wilkins 88] is a hierarchical non-linear planning system allowing some resource reasoning and replanning. Planning and execution are interleaved by partitioning the set of goals: the user specifies which goals can be delayed. An original plan is elaborated for the non-delayed goals.

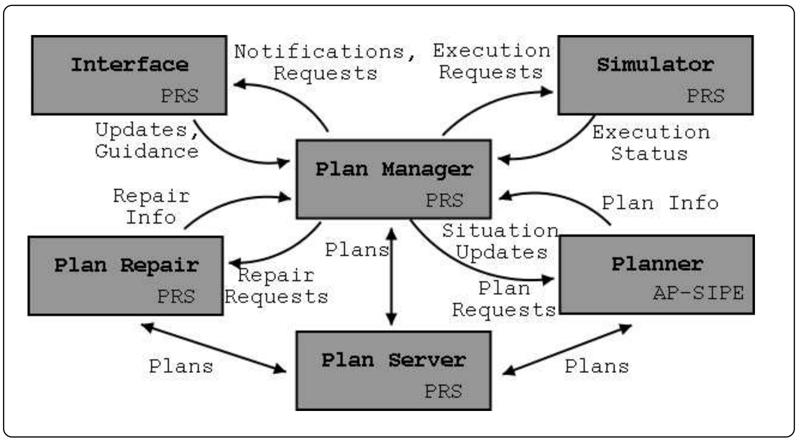


Figure 1.4: *The CPEF architecture, from [Myers 99]*

Its execution is started in parallel with a background planning process for the delayed goals. The original plan is executed until either an unexpected event occurs or the goals whose planning has been delayed are reached. The "background" plan is then retrieved (eventually after waiting for the planning process to finish) and updated with information on already executed actions. In case of unexpected situation, SIPE also offers a replanning ability, which corresponds in fact to a **plan repair**: *"transform the plan, retaining as much of the old plan as is reasonable, into one that will still accomplish the original goal from the current situation"*. The unexpected situation is translated into a list of problems in the plan. A set of domain-independent replanning actions is then used to modify the plan (by inserting new goals). This new planning problem is finally solved by the standard planner.

SIPE has been integrated into a *Continuous Planning and Execution Framework* (CPEF) [Myers 99] to update plans in a timely fashion in response to new information and requirements. CPEF proposes a distributed multi-agent architecture (cf. Figure 1.4) in which plan generation, execution, monitoring and dynamic plan repair are fluidly integrated. The SIPE-2 *planner* provides plan generation and adaptation capabilities. Users can influence the planning process through the *Advisable Planner* (AP). Plan execution and monitoring are performed by the procedural system PRS [Georgeff 89]. The always active *plan manager* monitors the environment and directs the overall plan generation, monitoring and execution processes. CPEF supports indirect execution, i.e. supervises activities execution by agents outside the system. The plan execution tracking is limited by the fact that reports on outcome of actions are taken into account in precedence order on actions. Finally, an interesting feature of CPEF is the specification of failure models and various monitors (as event-response rules). Failure models include unattributable failures and aggregate failures. *Failure monitors* encode appropriate responses to failures. *Knowledge monitors* test for the availability of information needed for decision making, and *Assumption monitors* test for situation changes that violate plan assumptions. This architecture has been tested by simulating realistic air-campaigns but has not been interfaced with a

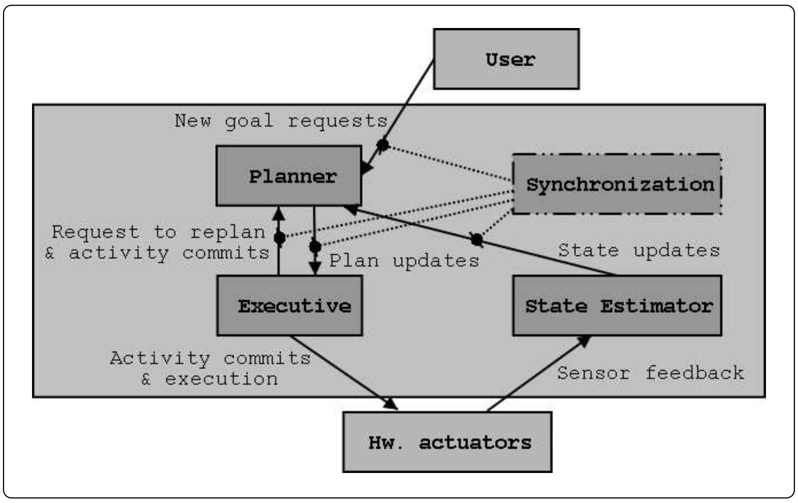


Figure 1.5: *The CASPER architecture, from [Chien 00]*

"functional level" of an autonomous agent.

IPEM and SPEEDY are purely classical planning systems and time is not represented explicitly in SIPE (time is managed as a "consumable resource"). CASPER [Chien 00] extends an IPEM-like approach to metric time and resources. In the CASPER system (*Continuous Activity Scheduling Planning Execution and Replanning*), the ASPEN planner [Rabideau 99] is provided with a set of goals, a plan, the current state and projections of the state in the future according to the plan. This plan is extended incrementally and regularly updated to respond to feedback. The planner receives and propagates updates: state updates (corresponding to changes to the state variables), activity updates (notifications of activity start and end times) and resource updates (actual resource usages and availabilities). Potential future conflicts are incrementally resolved by performing iterative repair techniques with a most-commitment strategy (all parameters are instantiated). This strategy simplifies the projections, especially the estimation of the resource level profiles, but reduces the plan flexibility.

Some issues relevant to interleaved temporal plan execution and repair are raised in [Chien 00], namely: *"When to replan? What to execute during planning time? How much time should the planner be given to replan? How to ensure the planner does not change activities that are already in execution?"*

The CASPER approach attempts to minimize the amount of time for replanning. Replanning is done when the projection of the current state according to the current plan reveals future flaws. During replanning, and depending on the domain application, nothing is executed (rover application) or a portion of the old plan is executed (spacecraft application). In that case, a commitment window, corresponding to the replanning time interval forbids the modification of committed activities by the replanning process, as well as the execution of non committed activities by the executive. Note however that

CASPER cannot handle conflicts which appear within the commitment window.

Figure 1.5 summarizes the CASPER basic mechanisms. The synchronization process sequences the tasks performed by the planner: planning, goal update, state update, executed plan update and commitment status of activities update. CASPER has been integrated in the Rocky7 rover architecture for scientific goals planning and has been tested on-board the EO1 satellite as part of the ST6 technology demonstration.

Interleaved scheduling and execution processes have also been used to improve the management of the operations in ground control centers. The approach proposed in the TIMELINE software [Grandjean 04] and implemented by EADS-Astrium in the INTELSAT control center, integrates operations scheduling (including resource allocations such as antennas ...), execution monitoring and re-scheduling (required by a task or constraint modification induced by the user interface or by the execution process). TIMELINE relies on a multi-threaded software architecture. A *schedule management thread* generates, when required, “scheduling jobs” which are handled by a *scheduling thread*. These jobs are ranked according to an emergency priority and processed one by one. Their results are applied to the current schedule. As for the CASPER system, a commitment window (corresponding to an upper-bound of the scheduling possible duration) forbids the modification of the operations occuring during this time interval. The scheduling process relies on classical constraint propagation algorithms. Reactivity is improved by dedicated algorithms and by limiting “scheduling jobs” to re-optimize only a portion of the global schedule.

Many systems, confronted with “real” applications and requiring reactivity, supply interleaving mechanisms between planning and plan execution processes. Various techniques have been applied so far, but very few really take into account timing problems, such as actions with durations and goals with deadlines, and their effects on both processes.

1.3.5 Planning at any level of abstraction

Opposed to the approaches presented in the previous sections, where planners are mainly used to schedule high-level actions and optimize resource usage over a long-term mission horizon, a new architectural concept, called IDEA (*Intelligent Distributed Execution Architecture*), is currently under development at NASA Ames Research Center. The IDEA approach [Muscettola 02] proposes to use planning as the core reasoning engine at each level of abstraction: mission planning as well as reactive execution. The basic concept behind IDEA is that any control system can be designed as a plan database and a planning mechanism to select the next action to execute in response to new goals and new sensory inputs.

IDEA defines a unified virtual machine. Complex autonomous systems can be designed as networks of IDEA agents interacting through a uniform, declarative communication protocol. The IDEA “virtual machine” consists of four main components (cf. Figure 1.6).

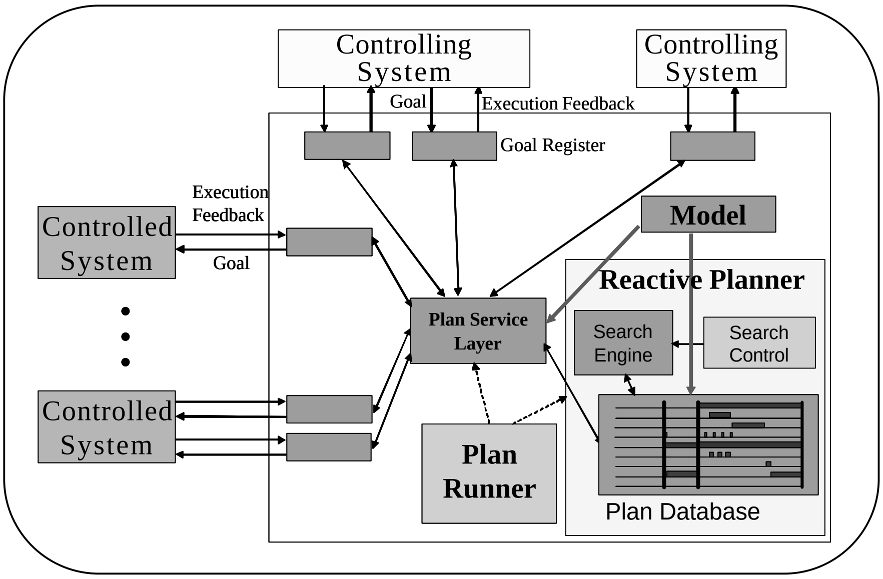


Figure 1.6: *Structure of an IDEA agent, from [Dias 03]*

A *plan database* represents the actions currently executed as well as the actions planned for the future. A *plan runner* is in charge of closing the loop by integrating execution feedback and reading the next actions to execute in the plan. The *plan service layer* provides the communication infrastructure for actions, goals and sensor data exchanges with other controlled or controlling agents and/or hardware drivers. Finally, a *reactive planner* is activated by the plan runner each time a new information is received or an action deadline expires.

The plan database is composed of a set of parallel timelines. Each timeline corresponds to a specific state variable (representing a dynamic property of the system), and describes the sequence of (past and future) actions and states linked to this state variable. Both actions and states are represented by the same structure, called *token*. A token corresponds to a time interval during which a procedure has to be executed. This procedure may have a set of input values and a vector of return status parameters. It is sent to other agents or drivers for execution. A procedure is completed when the return status is bound to a specific value or when the token deadline has elapsed.

Each timeline should contain one active token at current time. Thus, when a procedure returns or must be terminated, the reactive planner uses subgoaling search and constraint satisfaction techniques to guarantee that the next tokens are consistent with respect to the constraints in the agent's model. Reactive performance is obtained by limiting the horizon over which planning operates (e.g. make decisions only for the next step).

Each agent is given an *execution latency* that corresponds to the duration allowed for the "sense/plan/act" cycle to complete. This latency permits a quantification of the agent's responsiveness (the maximal duration between an event and its response is always equal to two latencies). Deliberative planning can also be introduced as a module controlled by the plan runner that operates on the same plan database but on a planning horizon that does not intersect with the reactive planning horizon.

To determine if an executive based on model-based reactive planning can be as responsive as, for instance, a procedural executive, some experimentations have been carried out to control a rover and an outdoor robot. During two stays[4] at NASA Ames, I took part[5] in the elaboration of a rover executive based on model-based reactive planning. Two IDEA agents (a system-level or "executive" agent and a mission-level agent) have been designed and implemented to autonomously control the K9 rover in real-time. The system has been evaluated in the scenario where the rover must acquire images from a specified set of locations. The IDEA agents are responsible for enabling the rover to achieve its goals while monitoring the execution and safety of the rover and recovering from dangerous states when necessary. The system-level agent achieved a 2Hz control rate on a 300MHz Pentium. The implementation of the two interacting IDEA agents and the models used to control the K9 rover are further detailed in [Dias 03].

This preliminary experimentation has been extended to control an outdoor robot (Gromit, an RWI ATRV Junior) embedding more complex functionalities requiring synchronization between processes and unary resource allocation. The robot navigation is based on a map of the environment computed by fusing position-tagged stereo-correlated images, while the cameras are also used to monitor en route interesting targets. An interesting feature of IDEA is the possibility to assign various planning horizons to the reactive planner. Different models have been tested on Gromit (see [Finzi 04]). A purely reactive behavior (planning one latency ahead) requires a careful (and somewhat painful) model design, with mixed domain and control knowledge: for each execution context, the set of available procedures is explicitly specified. This model achieved a control of Gromit with a 0.3s latency. Since loops appeared in the behavior of the functional processes, a model reflecting one loop and functioning with a "long-term" reactive horizon (40s) has been tested. The reactive planner plans for this intermediate horizon and then monitors the execution of this plan at each execution cycle. This look-ahead makes the model design a lot easier but decreases the performance (1.5s latency).

The planning technique currently used in the IDEA framework does not handle non-unary resources yet, but the look-ahead abilities should allow the management of resource levels on a long-term horizon. The plan repair possibilities are also limited. When operating with some look-ahead, if the return status of procedures are not consistent with the plan expectations, the plan is broken and has to be regenerated. Compared with a procedural executive, an IDEA executive is somewhat slower but acceptable. Its main advantages are the look-ahead abilities with flexible planning horizons; the use of a formal model easier to validate; and the use of a common language at any level of abstraction.

[4] In summer 2001 and summer 2002.

[5] This work has been done in collaboration with M. B. Dias, PhD-student at Carnegie Mellon University, under the direction of N. Muscettola and with the help of the "IDEA team": G. A. Dorais, C. Fry, R. Levinson, C. Plaunt and B. Vijayakumar.

1.3.6 Conclusion

We emphasized in this section the need for an organization of the software components in a well-defined architecture. Robust mission execution especially requires a decisional level with planning capabilities and smart execution mechanisms. A procedural executive provides task refinement and reactive exception handlers. Planning look-ahead capabilities are still required to manage unforeseen situations and resource levels over the mission duration. Interaction between both processes is a key problem. Batch planning implies long idle times. A more reactive behavior, interleaving planning and plan execution amounts to a trade-off between search completeness and reactivity, indeed plan execution commits plan steps early but gathers information that may open or prune alternatives for the planner. Likewise, interleaving plan repair and plan execution also requires a compromise between reactivity and plan quality: the longer the planner is given, the more likely it will be able to resolve all of the problems.

Finally, very few approaches explicitly deal with timing problems. We are concerned with applications requiring durative actions and goal deadlines. Thus we advocate the use of a **temporal executive** that can decide on action start times (and eventually end times), monitor action durations and rendez-vous and react to different types of "timing failures". Dealing with time at execution time is delicate: the execution process itself takes some time (in [Tsamardinos 98], the authors propose an efficient execution strategy for temporal plans based on STN techniques). The issue becomes even more difficult if temporal execution is combined with plan adaptation. The temporal executive has then also to control the planning process duration. A notion of execution cycle appears: update the state knowledge or integrate messages, plan if necessary, execute the actions on time. This cycle has a certain duration that has to be taken into account in the temporal plan model and search (cf. the update rate and commitment window in CASPER and the execution latency in IDEA).

1.4 Our approach

In IxTeT-eXeC, we propose a framework to combine temporal -deliberative planning, -plan repair and -execution control. Figure 1.7 summarizes our approach. The IxTeT-eXeC system is composed of two interacting components: the IxTeT planner that provides deliberative mechanisms and a new temporal executive TeXeC. IxTeT-eXeC has been integrated in the decisional level of the LAAS architecture [Alami 98], in interaction with the procedural executive OpenPRS [Georgeff 89].

IxTeT-eXeC provides two mechanisms to react to events (failures or new goals): (1) concurrent plan repair and plan execution for reactivity, and (2) replanning to regain completeness properties (providing there is enough time).

1. If some temporal flexibility remains, **plan repair** occurs after a partial invalidation of the plan. The plan structure is kept and the planner tries to restore the lost

properties. Execution of the valid part of the plan is pursued in parallel.

2. **Replanning** consists in abandoning the failed plan and generating, from the current situation, a new one for the remaining goals.

Both plan repair and replanning are based on the same lifted partial-order planning techniques, modified to take into account resource updates and time constraints. Indeed, we want our approach to address the following issues:

- the plan is regularly updated (including resource levels) and checked,
- the system still reacts to events even during plan repair,
- execution takes time into account,
- plan repair and replanning take time and planning duration into account.

It should be noted however that our work focuses on the robust execution of a mission plan with temporal and resource constraints and that the proposed approach presents some limitations. The non-linear CSP-based planning technique has interesting features w.r.t plan execution (flexibility, plan adaptations), but its performances somewhat limit the size of the planning problems that can be handled. It is also difficult to produce plans optimized according to a specific criterion (e.g. some resource consumption, the makespan, etc.)[6]. The performances of IxTeT-eXeC also depend on the flexibility left in the plan. If failures occur during the execution of a temporally over-constrained plan, the replanning process may abandon most of the goals. In any case, if the situation becomes critical, one can always rely on predefined emergency plans and procedures to put the system in a safe state.

[6] The plans are least-committed in the number of actions and constraints.

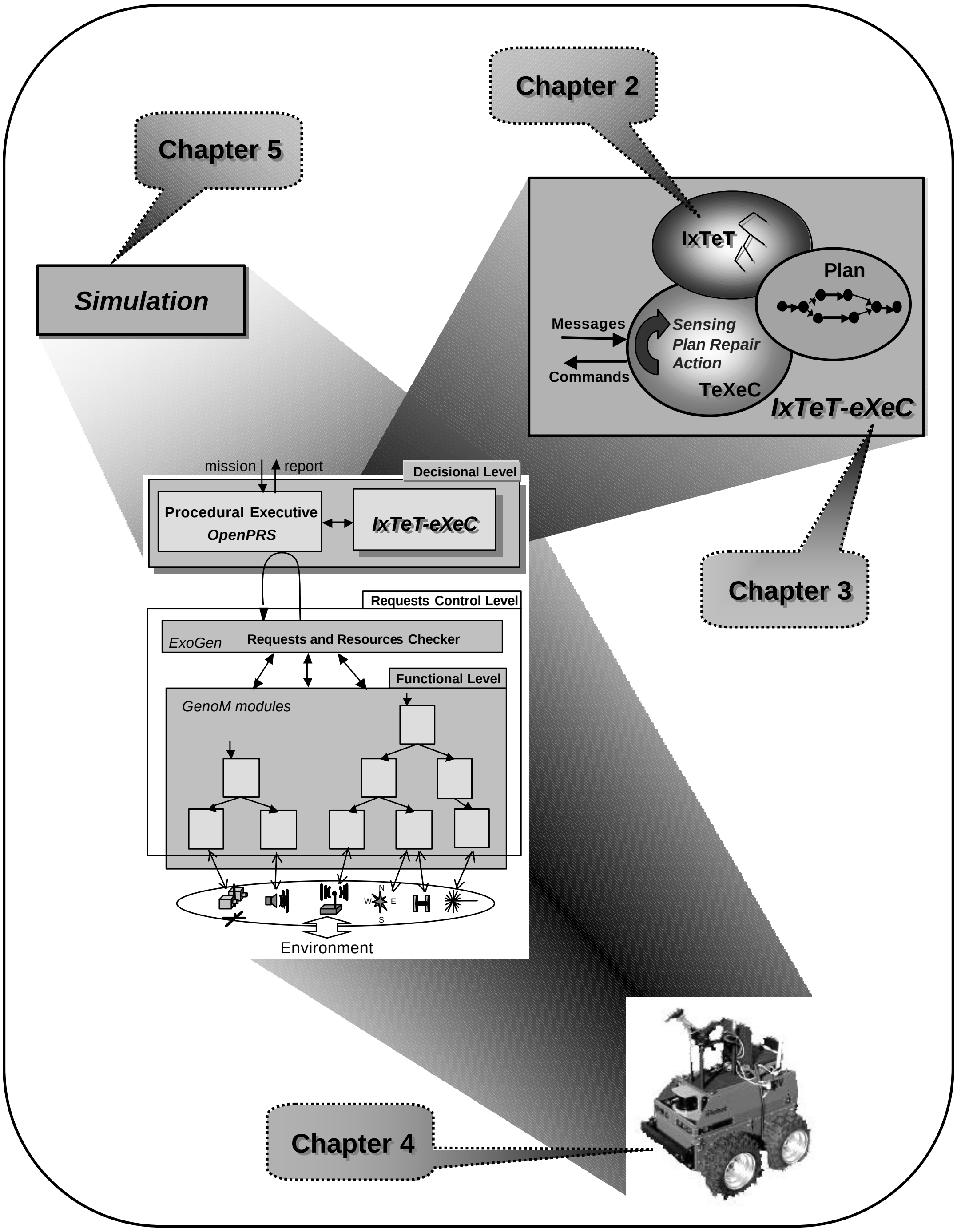


Figure 1.7: *Outline*

1.5 Mission example

The evaluation of our planning and execution architecture requires attention to the experimental setup and to the definition of reasonable metrics of performance. In fact, it is not possible to access standard benchmarks (such as those used by the planning community [Fox 02a]) for our domains of interest because they do not involve a realistic treatment of time and resources. Our objective is to demonstrate and evaluate our system in a realistic environment with relevance to space applications.

Thus we decided to integrate IXTET-EXEC in the LAAS architecture and perform tests on a real robotic platform. We use the outdoor robot Dala[7] as a "simulated" planetary rover with an exploration mission scenario. A navigation task in a totally unknown environment implies complex processes such as localization, map building or motion generation. Many contributions on these issues have been developed at LAAS [Lacroix 02] and integrated in the LAAS architecture.

We used these autonomous navigation capabilities to build up an exploration mission scenario. The rover receives three types of goals. A high-priority goal requires the rover to be back at the lander location by the end of the mission horizon. Medium-priority goals request communications between the rover and an orbiter or a ground-station during its visibility windows. Finally, lower-priority goals consist in a list of targets to take a picture of. At the mission planning level, the **planetary rover domain** is composed of the following set of actions:

- move($?x_1, ?y_1, ?x_2, ?y_2$) - This action represents the navigation task between two locations. The coordinates ($?x_i, ?y_i$) correspond to real numbers, the origin (0,0) being defined as the start location (near the lander). The actions move-x and move-y are possibly added, respectively corresponding to a navigation along the x-axis or the y-axis. A navigation task involves map building and stereo odometry, thus the cameras are required to look forward during a move action. Given the speed of the rover and the distance between the two locations, an estimate of the action duration can be computed. Note however that this estimate cannot take into account in advance the time needed to eventually avoid obstacles and adjustments need to be done during the plan execution.
- take-image($?target, ?x, ?y$) - The rover takes a high-quality image of a specific target at location ($?x, ?y$). This action requires that the rover stays still and that the cameras look towards the target. The image is compressed and stored. Once again, the model cannot be accurate: the compression rate varies according to the image that is actually taken, and the memory storage real availability needs to be updated during the plan execution.
- move-pan-tilt-unit($?pos_1, ?pos_2$) - This action changes the orientation of the cameras.

[7] Dala is an iRobot ATRV equipped with odometry sensors and a stereo camera pair mounted on a pan&tilt unit.

- download-images() - This action downlinks the stored images to the lander and thus frees some memory storage. Likewise, the quantitative effects and duration of this action are not precisely known in advance.

- communicate(?w) - The rover is given a prediction of the visibility windows. This action corresponds to the communication between the rover and the ground-station during the visibility window ?w. It is not compatible with the action download-images (they use the same channel resource) and requires the rover to stay still. Note that this type of action enforces temporal rendez-vous with the ground station.

- init-PTU-driver() and init-mvt-generation() - These actions are failure handlers. They correspond to the initialization of some functional processes in response to a certain type of failure.

A real planetary rover would also require a careful monitoring and management of the battery level. Since the robot we are using is not equipped with autonomous means to provide its energy (no solar panels for instance), we did not take this resource into account in the model described above. However, the IxTeT-eXeC system can handle problems where resource usages during actions can be estimated but not accurately predicted, and where the resource level should be regularly checked and projected in the future.

Figure 1.7 summarizes the outline of this thesis. Chapter 2 describes the underlying IxTeT planning system. We focus especially on its interesting features in a context of plan execution (representation of time and resource, flexibility, plan space search), and present the extension that we[8] made to provide a more realistic and flexible management of resources. Chapter 3 gives a detailed description of the IxTeT-eXeC framework and formally addresses the issue raised by the interleaving of plan repair and execution processes on the same plan. Chapter 4 presents the integration of IxTeT-eXeC in the LAAS architecture and the experiments carried out on a robotic platform. Finally, chapter 5 presents additional examples and results obtained by simulation[9].

We will use this planetary rover domain to illustrate Chapters 2 and 3.

[8] Part of this extension has been done jointly with Romain Trinquart.

[9] The controlled system is simulated by specific procedures in OpenPRS.

Chapitre 2

Ce chapitre décrit le système de planification IxTeT. Ce planificateur a été développé au LAAS et a fait l'objet de nombreuses contributions : recherche dans l'espace des plans partiels avec une stratégie de moindre engagement [Ghallab 94, Laruelle 94], intégration de la gestion de ressources partageables [Laborie 95b, Laborie 95a], hiérarchie d'abstraction gérée dynamiquement pendant la planification [Garcia 95] et extension des gestionnaires de contraintes pour traiter des variables atemporelles numériques et des contraintes mixtes entre variables temporelles et atemporelles [Trinquart 01].

La première section présente la représentation fonctionnelle à base de chroniques utilisée pour modéliser les opérateurs et le problème de planification. La deuxième section détaille les reseaux de contraintes et les techniques de satisfaction de contraintes employées. La troisième section décrit l'algorithme de recherche dans l'espace des plans patiels. Finalement, la dernière section détaille notre contribution : une extension de l'expressivité et de la flexibilité dans la gestion des ressources.

Une représentation fonctionnelle à base de chroniques

Variables d'état

Le monde est décrit par un ensemble de variables d'état (ou attributs logiques) qui sont des fonctions du temps multivaluées. On distingue des attributs contrôlables, contingents et rigides.

"Timeline"

Le temps est représenté par un ensemble ordonné d'instants (ou *timepoints*) qui sont des variables sur $\Re^+$. L'évolution de la valeur des attributs est spécifiée par la proposition *hold*, qui impose la persistance d'une valeur sur un intervalle temporel, et la proposition *event* qui représente un changement instantané de ressource. A chaque attribut logique est associée une *timeline* qui représente l'historique de ces changements et persistances.

Ressources

D'autres attributs représentent spécifiquement des types de ressource. Les propositions *use*, *consume* et *produce* spécifient, respectivement, l'emprunt sur un intervalle, la consommation ou la production à un instant donné d'une certaine quantité de ressource.

Représentation des actions

Une action comporte : un ensemble d'*events* décrivant les changements du monde induits par l'action, un ensemble d'assertions *hold* exprimant les conditions requises ou la protection d'un fait entre deux événements, un ensemble de propositions d'utilisation de ressources, un ensemble de contraintes temporelles entre les timepoints de l'action et un ensemble de contraintes entre les variables atemporelles.

Problème de planification

Le problème de planification est également décrit par une chronique (i.e. un ensemble de propositions temporelles et de contraintes) qui comprend : l'état initial (les valeurs initiales des attributs instanciés), les événements prévus pour certains attributs contingents, les profils de disponibilité des ressources prévus, les buts à accomplir (valeurs souhaitées pour des attributs instanciés spécifiques) et un ensemble de contraintes entre ces éléments.

Discussion concernant l'exécution du plan

Dans cette partie, nous discutons l'influence de l'exécution sur la représentation des actions et la définition des modéles. Des informations supplémentaires sont nécessaires (interruptibilité des actions) et certaines précautions doivent être prises.

Les gestionnaires de contraintes sous-jacents

Pendant la recherche, l'affinement du plan ajoute des contraintes : contraintes de précédence entre timepoints, contraintes d'unification entre variables atemporelles, contraintes numériques sur les quantités de ressource, etc. Ces contraintes sont gérées grâce à des techniques de satisfaction de contraintes (un CSP temporel et un CSP atemporel). Un CSP (*Constraint Satisfaction Problem*) est défini par un ensemble de variables, leurs domaines respectifs et un ensemble de contraintes entre ces variables. Lors de la planification, des algorithmes de propagation calculent les domaines minimaux qui garantissent la consistance des CSPs.

Le réseau temporel

Il est modélisé par un STN (*Simple Temporal Network*) (les variables sont les timepoints et les contraintes sont des intervalles numériques représentant la durée entre deux timepoints). Des algorithmes de type Floyd-Warshall assurent la propagation.

Le réseau atemporel

Ce CSP gère des variables atemporelles (arguments, valeur des attributs, etc.) symboliques et numériques, avec des contraintes de type unification, séparation, restriction de domaine, numériques, etc. Deux types de techniques de propagation sont employées : une technique exhaustive et coûteuse qui assure la consistance globale du réseau, et une technique de filtrage qui assure sa consistance locale.

Les contraintes mixtes

Des contraintes mixtes entre variables temporelles et atemporelles peuvent être spécifiées: $?x = c * (t_j - t_i)$. Elles sont gérées par un superviseur qui transfère les informations d'un CSP à l'autre.

Discussion concernant l'exécution du plan

L'utilisation de techniques de satisfaction de contraintes offre deux avantages principaux: une modélisation plus réaliste du monde (notamment concernant les effets quantitatifs des actions) et une flexibilité du plan produit.

Planification non linéaire causale avec des variables partiellement instanciées

La planification consiste à déterminer, à partir d'un modèle de la situation initiale et des actions réalisables, un plan d'actions qui permet d'accomplir un ensemble de buts. IxTeT effectue une recherche dans l'espace des plans partiels : un plan incomplet est progressivement affiné par l'insertion d'actions et de contraintes jusqu'à l'obtention d'un plan solution valide. Un noeud de l'arbre de recherche est alors un plan partiel et une transition entre deux plans partiels correspond à une transformation du plan pour résoudre un défaut. Par ailleurs, les plans produits sont partiellement ordonnés et instanciés et des liens causaux permettent de conserver les liens entre les actions (protéger l'établissement d'une propriété par une action).

Cette section présente l'espace de recherche et l'algorithme de planification dans le cadre d'IxTeT, détaille les défauts et résolvantes potentiels et discute les heuristiques utilisées pour guider la recherche.

Recherche dans l'espace des plans partiels

Un noeud de l'arbre de recherche est un plan partiel caractérisé par (A, C, L, F), i.e. un ensemble d'actions A, un ensemble de contraintes C sur les variables apparaissant dans A et L, un ensemble de liens causaux L et un ensemble de défauts F. En utilisant la représentation d'IxTeT, un plan peut être représenté de façon similaire par une chronique $(Evt, Hold, Use, Prod, Cons, C)$ qui contient des ensembles de propositions *event*, *hold*, *use*, *produce*, *consume* et les contraintes sur les variables apparaissant dans ces propositions.

Une telle chronique peut contenir trois types de défauts :

- des *sous-buts*, i.e. des propositions *event* ou *hold* pas encore établies (la valeur de l'attribut n'est pas justifiée par l'état initial ou par un événement antérieur);
- des *menaces*, i.e. des paires de propositions *event* et *hold* dont les valeurs sont potentiellement en conflit (un attribut instancié ne peut avoir qu'une seule valeur à un instant donné);
- des *conflits de ressource*, i.e. des ensembles de propositions *use* qui sont potentiellement en recouvrement et qui surconsomment une même ressource.

Un plan partiel est un plan solution si il ne contient aucun défaut et si les réseaux de contraintes sous-jacents sont consistants.

Une étape de planification dans IxTeT

Une étape de planification se décompose en quatre phases :

1. le plan partiel est analysé pour détecter ses défauts et associer à chaque défaut une disjonction de résolvantes potentielles,
2. un défaut est sélectionné,
3. une de ses résolvantes est choisie,
4. cette résolvante est insérée dans le plan.

Si le plan contient un défaut nécessaire ou si la résolvante aboutit à un plan inconsistant, l'algorithme revient sur les points de choix des résolvantes.

Analyse des défauts

L'analyse recherche les défauts, calcule les résolvantes et associe un coût à chaque résolvante qui sera utilisé pour guider la sélection du défaut et le choix de la résolvante à appliquer. Ce coût est calculé en se basant sur une stratégie de moindre engagement. Une telle stratégie implique de choisir le défaut avec le facteur de branchement le plus faible (le moins de résolvantes). Elle a été étendue dans IxTeT pour prendre en compte l'engagement de chaque résolvante : le coût d'une résolvante reflète son influence sur la réduction de l'espace des plans solutions atteignables depuis le plan partiel courant.

Cette sous-section détaille pour les trois types de défauts : comment ils sont détectés, leurs résolvantes et leurs fonctions de coût.

Contrôle de la recherche

Les défauts du plan partiel sont classés en fonction d'une hiérarchie d'abstraction. Le planificateur traite tous les défauts appartenant à un même niveau d'abstraction avant de prendre en compte les défauts du niveau suivant. La hiérarchie définie dans IxTeT est associée aux noms des attributs et vérifie la propriété de Monotonicité Ordonnée. Elle diffère des autres hiérarchies par sa flexibilité : un graphe d'abstraction est calculé hors-ligne, mais les niveaux sont définis en ligne, en fonction de l'évolution de la recherche.

Parmi les défauts de l'état d'abstraction courant, un défaut est sélectionné en fonction d'un facteur (K) calculé à partir des fonctions de coût de la disjonction de résolvantes associée. Ce facteur suit une stratégie de moindre engagement opportuniste.

Enfin, la progression dans l'arbre de recherche peut être guidée de deux façons différentes: un algorithme A^ϵ, ou une recherche en profondeur ordonnée d'abord. Les fonctions heuristiques $f = g + h$ sont calculées en fonction des coûts des résolvantes.

Nous discutons finalement les performances globales du planificateur et les avantages et inconvénients des diverses heuristiques.

Discussion concernant l'exécution du plan

La recherche dans l'espace des plans partiels fournit un bon cadre pour la planification incrémentale et la réparation de plan.

Extension de la gestion des ressources

Nous avons étendu la représentation des ressources dans deux directions:

- une ressource type peut maintenant avoir plusieurs allocations possibles,
- les quantités empruntées, consommées ou produites pendant une action peuvent être représentées comme des variables numériques.

Conflit de ressource

Ces modifications imposent de redéfinir la notion de conflit de ressource.

Détection des conflits

De même, les algorithmes de détection (recherche de cliques surconsommatrices dans un graphe des intersections possibles) doivent être modifiées.

Résolvantes

Finalement, il faut définir de nouveaux types de résolvantes. En distinguant plusieurs types de conflits de ressource, on peut de plus réduire le facteur de branchement.

Cette extension des ressources permet une représentation plus réaliste du monde, mais surtout augmente la flexibilité d'un plan vis à vis de l'utilisation des ressources, le rendant plus robuste aux aléas de l'exécution.

Chapter 2

The IxTeT planning system

As introduced in the previous chapter, embedding decisional capabilities in autonomous space systems strongly suggests the use of a planning system capable of reasoning about time and resources. In a context of plan execution in a dynamic environment, the IxTeT planner/scheduler has interesting properties. Based on a time-oriented representation, it combines a POCL[1] planning process with constraint satisfaction techniques, thus producing flexible and parallel plans. Moreover, previous works (notably IPEM [Ambros-Ingerson 88]) have demonstrated that, in domains without temporal and resource considerations, a plan-space search process can be adapted to perform interleaved planning and plan execution.

In this chapter, we describe the IxTeT planning system. Developed at LAAS, it originally proposed an efficient Time Map manager whose lattice of timepoints relied on an indexed spanning tree (IxTeT stands for *Indexed Time Table*) [Ghallab 89]. It has been improved by multiple contributions: search in the plan space with a least commitment strategy [Ghallab 94, Laruelle 94], integration of sharable resource management [Laborie 95b, Laborie 95a] and integration of abstraction hierarchies dynamically managed during planning [Garcia 95]. Still, IxTeT exhibited a weak expressiveness in a context of real-world applications. The atemporal CSP manager could only handle symbolic variables ranging over finite domains. It was not possible for instance to represent the initial and final locations of a move action by real numeric coordinates. Besides, the resource manager could only handle resource usages of a constant numeric quantity, thus forbidding the expression of more realistic relations between the effects of an action and its parameters (the energy consumed by a $\mathsf{move}(?loc_1, ?loc_2)$ action, for example, varies with the distance between $?loc_1$ and $?loc_2$).

A recent work [Trinquart 01] has improved the expressiveness of the planner by extending the CSP manager of atemporal variables to handle mixed symbolic and numeric variables, ranging over finite and continuous domains. Has also been added the possibility to express mixed constraints between temporal and atemporal variables, thus allowing the representation of dependence between the effects of an action and its duration (typically the duration of a $\mathsf{move}(?loc_1, ?loc_2)$ action varies with the speed of the rover and the

[1] Partial Order Causal Link

distance between $?loc_1$ and $?loc_2$).

One goal of this chapter is to provide the reader (and the potential IxTeT user) with a unified picture of all aspects of the current version of the planner, gathering information essentially from [Laruelle 94], [Laborie 95a], [Gaborit 96], and [Trinquart 04].

The last part of the chapter focuses on our main contribution to the IxTeT planner: we have proposed and implemented algorithms to extend the resource management to deal with allocated resources and the usage of variable quantities of resources. Other "minor" modifications resulted from our tests of IxTeT-eXeC with the rover application and the need to improve the performances.[2]

This chapter is organized as follows. Section 2.1 presents the time-oriented functional representation. Section 2.2 details the underlying constraint networks and constraint satisfaction techniques. Section 2.3 describes the search algorithm in the plan space. Finally, section 2.4 details our contribution in the resource management.

2.1 A time-oriented functional representation

The IxTeT formalism presupposes that the world is fully observable and that the actions are deterministic. The world is described by a set of state variables, each being a function of time. During planning, the evolution of the value of the state variables is partially specified by temporal propositions picturing change and persistence. For each state variable, a timeline represents its history of changes and persistences over time.

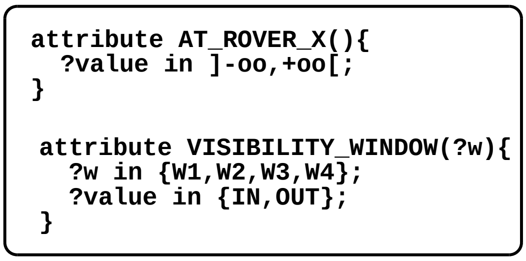
2.1.1 State variables

State variables are also called **logical attributes** in the IxTeT formalism. A logical attribute stands for a property of the world that can take only one value at a time (e.g. the position of the rover). It is defined by the tuple:

$$(AttName, (?x_1, \ldots, ?x_n), ?v_t)$$
$$\text{(also noted: ``}AttName(?x_1, \ldots, ?x_n) :?v_t\text{'')}$$

$AttName$ refers to the symbolic name of the function, $(?x_1, \ldots, ?x_n)$ to its arguments and $?v_t$ to its value at time t. The arguments are variables ranging over finite domains, whereas the value is a variable ranging over finite or continuous domains. An attribute is called **grounded** if all its arguments are instantiated.

[2] These modifications are described in more details along the chapter and notably concern: the use of an Ordered Depth First search strategy instead of the A_ϵ algorithm, the modification of the backtrack strategy by caching partial plans and the introduction of contingent attributes into the abstraction hierarchy.



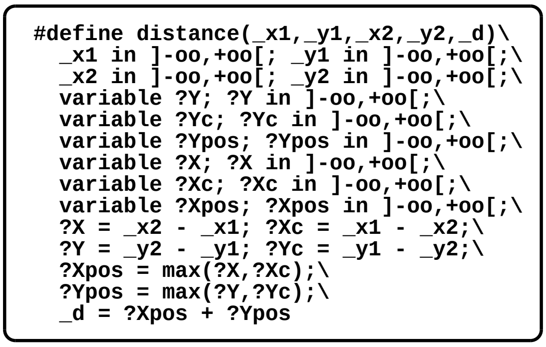
```
attribute AT_ROVER_X(){
  ?value in ]-oo,+oo[;
}

 attribute VISIBILITY_WINDOW(?w){
   ?w in {W1,W2,W3,W4};
   ?value in {IN,OUT};
 }
```

Figure 2.1: *Examples of logical attribute declaration*

Logical attributes can be separated into two classes: contingent and controllable attributes.

- The value of a **contingent attribute** varies with time, but cannot be controlled by the planning agent. An example is the attribute representing the visibility window $?w$ with the ground station: "$VISIBILITY\_WINDOW(?w)\ {:}?v$", with $?v \in \{IN, OUT\}$ (see Figure 2.1).
- The changes of value of a **controllable attribute** are controlled by the planning agent. Examples are the attributes representing the position of the rover along the x-axis and the y-axis "$AT\_ROVER\_X()\ {:}?x$" and "$AT\_ROVER\_Y()\ {:}?y$" with $?x, ?y \in \Re$ (see Figure 2.1).



```
#define distance(_x1,_y1,_x2,_y2,_d)\
  _x1 in ]-oo,+oo[; _y1 in ]-oo,+oo[;\
  _x2 in ]-oo,+oo[; _y2 in ]-oo,+oo[;\
  variable ?Y; ?Y in ]-oo,+oo[;\
  variable ?Yc; ?Yc in ]-oo,+oo[;\
  variable ?Ypos; ?Ypos in ]-oo,+oo[;\
  variable ?X; ?X in ]-oo,+oo[;\
  variable ?Xc; ?Xc in ]-oo,+oo[;\
  variable ?Xpos; ?Xpos in ]-oo,+oo[;\
  ?X = _x2 - _x1; ?Xc = _x1 - _x2;\
  ?Y = _y2 - _y1; ?Yc = _y1 - _y2;\
  ?Xpos = max(?X,?Xc);\
  ?Ypos = max(?Y,?Yc);\
  _d = ?Xpos + ?Ypos
```

Figure 2.2: *Example of constraints for the rigid property DISTANCE*

It is also possible to express **rigid** conditions, i.e. properties which value remains invariant in time. "$DISTANCE(?x_1, ?y_1, ?x_2, ?y_2)\ {:}?d$" for instance depicts the distance between the locations $(?x_1, ?y_1)$ and $(?x_2, ?y_2)$, or "$SPEED()\ {:}?s$" represents the speed of the rover $[0.03, 0.1]$. In IxTeT, these rigid properties are expressed and managed through a set of constraints on variables. Thus "$SPEED()\ {:}?s$" is equivalent to the constraint: "$?s$ in $[0.03, 0.1]$". Figure 2.2 illustrates the set of constraints for $DISTANCE$. It computes a manhattan distance by $?d = |?x_2 - ?x_1| + |?y_2 - ?y_1|$. As shown in the distance example, the type of rigid properties that can be expressed is limited by the type of constraints that are currently managed by the CSP managers. However, it is always possible to add specific constraints if one can define their propagation behavior.

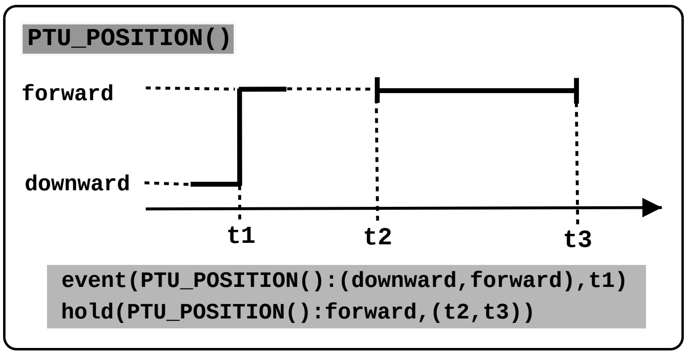

PTU_POSITION()
forward
downward
t1
t2
t3
event(PTU_POSITION():(downward,forward),t1)
hold(PTU_POSITION():forward,(t2,t3))

Figure 2.3: *Partial specification of the evolution of the attribute PTU_POSITION()*

2.1.2 Timelines

Time is explicitly represented as a set of linearly ordered instants. These instants, or **timepoints** are variables ranging over $\Re^+$. A time interval corresponds to the temporal distance between two timepoints. Binary constraints are expressed between two timepoints t_1 and t_2 such as ordering constraints $t_1 < t_2$ or $t_2 < t_1$, or more generally quantitative constraints: $(t_2 - t_1) \in I$, I being an interval on $\Re$.

Attributes are piece-wise constant functions of time. Their value evolution is temporally qualified by two types of temporal propositions:

- $hold(AttName(?x_1, \ldots, ?x_n) :?v, (t_1, t_2))$ asserts the persistence of the value $?v$ over the time interval $(t_2 - t_1)$ defined by the timepoints t_1 and t_2,
- $event(AttName(?x_1, \ldots, ?x_n) : (?v_1, ?v_2), t)$ states the instantaneous change of value from $?v_1$ to $?v_2$ at the instant t.

Figure 2.3 presents an example of partial specification of the evolution of an attribute representing the position of a pan&tilt unit.

2.1.3 Resources

A resource is defined as a substance or a set of objects whose availability induces constraints on the actions that use them. Whereas actions modify in an absolute way the value of state variables, a resource is only influenced in a relative way, by an increase or decrease of its available quantity.

Resources are classified according to their capacity and their usage.

- A **reusable** resource is used by an action and released when the action terminates. A device (such as a camera) is an example of **unary** reusable resource. It imposes a total order between the actions that borrow it. A multiple capacity reusable resource, such as the power of a battery, can be used by concurrent actions, providing that their cumulative usage does not exceed the resource capacity.

```
resource CAMERA(){
  defaultcapacity = 1;
}

reservoir resource MEMORY_STORAGE(?d){
  ?d in {DISK1,DISK2,DISK3,DISK4};
  defaultcapacity = 54660;
  capacity(DISK1) = 46675;
  capacity(DISK3) = 38894;
}
```

Figure 2.4: *Examples of resource attribute declaration*

- A **consumable** resource is consumed and possibly produced by actions. Especially, a **reservoir** resource, as for instance the space available for image storage, is a consumable resource with a maximum capacity. Planning with actions requiring a reservoir resource imposes to check that the level of the resource never exceeds the capacity or never becomes negative during the plan horizon.

In IxTeT, resources are gathered in a set of **resource types**: two resources belong to the same type if they can be equally used. Consider for instance that an image can be indifferently stored on disk1 or disk2, then both resources are of the same "memory storage" type. A resource type is represented by a **resource attribute**: $RsceType(?r)$ (e.g. $MEMORY\_STORAGE(?d)$, $?d \in \{disk1, disk2\}$). A resource is **allocated** when its parameter is instantiated.

In [Laborie 95b], the authors first introduced a resource management in the temporal planner IxTeT. They proposed algorithms to manage multiple capacity non-parameterized resources and resource usages of constant quantities. In section 2.4 we detail how we extended these algorithms to manage resource types with several possible allocations and resource usages of variable quantities.

More precisely, a resource is defined as shown in Figure 2.4 and each allocated resource is associated with a capacity[3] (a default one if the specific capacity is not mentioned).

During planning, the resource management algorithms guarantee that the level of a resource never becomes negative, but does not guarantee that it never exceeds the maximal capacity. Thus, when a resource attribute R is defined as a reservoir resource, it is internally associated with a complementary resource attribute R_CPT, which has a null capacity and a "mirror" usage: R_CPT is consumed when R is produced and vice versa.

Resource availability profiles and resource usages are described through three piecewise constant temporal propositions:

[3] Due to implementation constraints, the definition of the default capacity should correspond to the maximal capacity of the different allocated resources of the same type.

```
task TAKE_PICTURE(?obj,?x,?y)(t_start,t_end){
  ?obj in OBJECTS;
  ?x in ]-oo,+oo[; ?y in ]-oo,+oo[;

  hold(AT_ROBOT_X():?x,(t_start,t_end));
  hold(AT_ROBOT_Y():?y,(t_start,t_end));
  hold(PTU_POSITION():downward,(t_start,t_end));

  event(PICTURE(?obj,?x,?y):(none,doing),t_start);
  hold(PICTURE(?obj,?x,?y):doing,(t_start,t_end));
  event(PICTURE(?obj,?x,?y):(doing,done),t_end);

  use(CAMERA():1,(t_start,t_end));
  variable ?image_size;
  variable ?cr;
  compression_rate(?cr);
  ?image_size = 175610 * ?cr;
  consume(STORAGE():?image_size,t_start);

  (t_end - t_start) in ]0,60];
}nonPreemptive
```

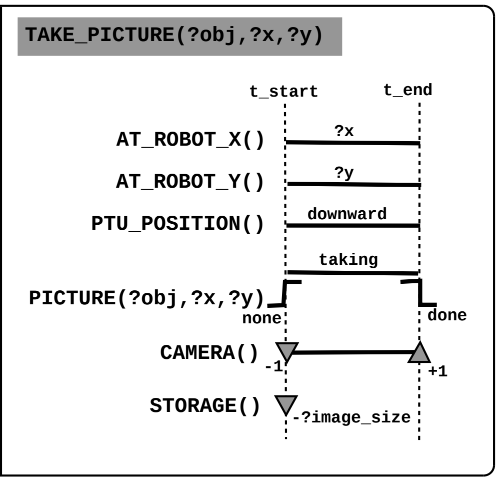


Figure 2.5: *Example of action declaration*

- *use*$(R(?r) :?q, (t_1, t_2))$ specifies the borrowing of a quantity $?q$ of the resource $?r$ of type R during the time interval defined by the timepoints t_1 and t_2,
- *consume*$(R(?r) :?q, t_c)$ specifies the consumption of a quantity $?q$ at time t_c,
- *produce*$(R(?r) :?q, t_p)$ specifies the production of a quantity $?q$ at time t_p.

$?q$ is a variable quantity whose domain is an interval of real numbers $[q_{min}, q_{max}]$.

2.1.4 Representation of actions

Using the terminology of [Ghallab 04], IxTeT follows a chronicle planning approach. A **chronicle** for a set of state variables is formally defined as a pair $(\mathcal{F}, \mathcal{C})$ where $\mathcal{F}$ is a set of temporal propositions, i.e. event and persistence conditions about the state variables, and $\mathcal{C}$ is a set of temporal and atemporal constraints.

In IxTeT, a chronicle on a set of logical attributes is represented by the tuple $(Evt, Hold, C)$ where Evt is a conjunction of *event* propositions on these attributes, $Hold$ is a conjunction of *hold* propositions and C is a conjunction of temporal, atemporal and mixed constraints between the variables (timepoints, arguments, values) appearing in Evt and $Hold$.

This structure is completed by taking into account resource attributes and resource temporal propositions. Thus, we define a chronicle on a set of logical attributes $LAtt$ and a set of resource attributes $RAtt$ as the tuple $(Evt, Hold, Use, Prod, Cons, C)$ where Evt and $Hold$ are respectively conjunctions of *event* and *hold* propositions on the attributes in $LAtt$; Use, $Prod$ and $Cons$ are respectively conjunctions of *use*, *produce* and *consume* propositions on the attributes in $RAtt$, and C is a conjunction of temporal, atemporal and mixed constraints between the variables appearing in Evt, $Hold$, Use, $Prod$ and $Cons$.

A planning operator, also called action or task, corresponds to a chronicle on subsets of logical and resource attributes. The *event* propositions describe the changes of the world induced by the action, the *hold* propositions express required conditions or the protection of some property between two events. Figure 2.5 shows an example of action description. To take a picture of a target $?obj$ at location $(?x, ?y)$, the rover must stay still at this location and the cameras should look downward throughout the action. The action borrows the camera pair and results in the storage of an image which size varies with the compression rate.

The user can also specify that a task A contains a set of subtasks $\{A_i\}$ whose starting and ending timepoints correspond to intermediate timepoints of A.[4]

The use of chronicles allows the definition of intermediate timepoints and thus the expression of more flexible and efficient interactions between actions (a resource can be released before the end of the action, intermediate effects can be used by concurrent actions, etc.).

2.1.5 Planning problem

The planner is given a planning domain and a planning problem. The planning domain consists in the declaration of logical and resource attributes, rigid properties and planning operators. The planning problem represents the initial scenario and the goals to achieve. In IxTeT, this problem is represented through a chronicle on logical and resource attributes.

The initial scenario describes the initial state of the domain: the initial value of each grounded logical attribute is established by an **explained**[5] *event* proposition, the initial level of each allocated resource (if different from the capacity) is set by a consumption of the difference between the capacity and the level. These initial propositions happen at the origin of the plan horizon. The initial scenario also describes the expected evolution of attributes independently of the actions, such as the evolution of contingent attributes[6] or the availability profiles of resources.

Goals are expressed as *event* or *hold* propositions that need to be established by the planning process. Two information are specified with each goal $(goal(p, d_{achiev}))$: a priority p and (if available) an estimation of the minimal duration needed to achieve the goal d_{achiev}. The problem chronicle also contains temporal constraints such as the duration of the planning horizon, deadlines on goal timepoints, etc. The example given

[4] However, this "task decomposition" is just a user programming facility to build models of complex tasks on the basis of simpler ones. After the compilation of the model into the data structure used by the planner, this task hierarchy is lost, the subtasks being replaced with their corresponding chronicles in the top-level task.

[5] The label **explained** before a proposition $hold(AttName(?x_1, \ldots, ?x_n) \; :?v*, (t, t*))$ or $event(AttName(?x_1, \ldots, ?x_n) \; : \; (?v, ?v*), t)$ is used to assert that the attribute has the value $?v*$ at time t, no matter what its previous value was; and the planner does not need to justify the establishment of this value. Thus the initial state is described through explained events occurring at the origin of the plan horizon. These events can be used to establish the values of future propositions.

[6] These temporal propositions are labeled as **contingent** in the initial task declaration.

```
task Init()(t_start,t_end){
    timepoint t_visi1, t_visi2;
    timepoint t_goal1, t_goal2s, t_goal2e;
    timepoint t_goal3s, t_goal3e, t_goal4s, t_goal4e;

    // Initial state
    explained event(AT_ROBOT_X():(?,0),t_start);
    explained event(AT_ROBOT_Y():(?,0),t_start);
    explained event(ROVER_STATUS():(?,still),t_start);
    explained event(PTU_POSITION():(?,forward),t_start);

    explained event(PTU_DRIVER_INITIALIZED():(?,true),t_start);
    explained event(MVT_GENERATION_INITIALIZED():(?,true),t_start);

    explained event(COMMUNICATION(W1):(?,none),t_start);
    variable ?x1,?y1;
    ?x1 in ]-oo,+oo[; ?y1 in ]-oo,+oo[;
    explained event(PICTURE(OBJ1,?x1,?y1):(?,none),t_start);
    variable ?x2,?y2;
    ?x2 in ]-oo,+oo[; ?y2 in ]-oo,+oo[;
    explained event(PICTURE(OBJ2,?x2,?y2):(?,none),t_start);

    // Visibility window
    contingent event(VISIBILITY_WINDOW(W1):(?,out),t_start);
    contingent event(VISIBILITY_WINDOW(W1):(out,in),t_visi1);
    contingent event(VISIBILITY_WINDOW(W1):(in,out),t_visi2);
    (t_visi2-t_visi1) in [120,120];
    (t_visi1-t_start) in [300,300];

    // Goals
    hold(AT_ROBOT_X():0,(t_goal1,t_end)) goal(3,0);
    hold(AT_ROBOT_Y():0,(t_goal1,t_end)) goal(3,0);

    hold(COMMUNICATION(W1):done,(t_goal2s, t_goal2e)) goal(2,0);

    hold(PICTURE(OBJ1,6,-3):done,(t_goal3s,t_goal3e)) goal(1,0);
    hold(PICTURE(OBJ2,9,8):done,(t_goal4s,t_goal4e)) goal(1,0);

    // Horizon
    (t_end - t_start) in [900,900];
}latePreemptive
```

Figure 2.6: *Example of planning problem*

in Figure 2.6 represents the planning problem for the planetary rover domain with a high priority goal to be back at the lander at the end of the mission horizon, a medium priority communication goal with the ground-station and two lower priority goals to take targets at different locations into picture.

2.1.6 Discussion with respect to plan execution

About the assumption on the determinism of actions

The assumption that planning operators are deterministic implies that the value of an attribute is not changed by the execution of an action, unless this change is explicitly declared. The rover evolves in a non-deterministic environment and, as a complex system, is subject to failures. However, the operator knows how the rover has been built and how it is expected to behave. Unlike the action of throwing a dice which has definitely a random effect, the non-determinism of actions in the rover domain comes from possible failures or external events. Thus, it seems reasonable to use a deterministic planning domain representation that models the nominal course of actions and let an execution controller check if an action execution is effectively nominal and react to failures.

Action execution

IxTeT-eXeC is composed of two components: the planning system and a temporal executive which interacts with an external procedural executive. The temporal executive is given a plan to execute, i.e. a set of partially ordered actions and decides when to start and eventually stop actions and monitors their timing, whereas the procedural executive expands actions into detailed commands to the controlled system, monitors the execution of each action and transmits status reports to the temporal executive upon action completion.

From an "execution point of view", an action in the plan is characterized by its name a, its grounded[7] parameters p_a, a starting timepoint st^a, an ending timepoint et^a and an identifier i_a. The temporal executive needs additional information on the action behavior. If an action is not controllable by the agent that can only wait for it to terminate, the action model is labeled *nonPreemptive*. If an action is controllable, it can be stopped as soon as possible (label *earlyPreemptive*) or as late as possible (label *latePreemptive*). Thus a move action is late preemptive: the rover can be stopped but is given as much time as possible to reach its destination. Considering the example of a robot that takes objects and moves them to diverse locations, the early preemptive carry action that monitors that the robot keeps an object in its hand is stopped as soon as the robot has reached the object destination.

Model design restrictions

The design of a domain involves two models: one (in the IxTeT formalism) for IxTeT-eXeC and one (a library of procedures) for the procedural executive. Actually, IxTeT-eXeC does not yet allow to exploit the full expressiveness of the IxTeT formalism and the following restrictions need to be applied.

The temporal executive launches actions by sending the information (a, p_a, i_a) to the procedural executive. The procedural model contains a procedure associated with each action name a. It describes the expansion into commands and a model of the report to send back to IxTeT-eXeC. This report contains a partial description of the system state, i.e. the procedural executive computes, from sensor information, the level of the resources used by the action and the current value for each logical attribute appearing in the action definition. Thus, when executing an action, the procedural executive needs to know the

[7] The parameters of an action are not necessarily grounded at the end of the planning process, in order to keep execution flexibility, but an action is completely instantiated before its execution: a value is chosen and propagated through the constraints of the plan.

values of the arguments of logical and resource attributes relevant to the action. *Thus, when designing the IxTeT domain model, one should be careful that all attributes are either grounded or their arguments are variables appearing in the list of the action parameters.*

During an action execution, an active monitoring of the resource levels is performed by the procedural executive[8]. As far as IxTeT-eXeC is concerned, the "theoretical" level of a resource in the plan is updated by its actual value at the end of each action requiring the resource and projected to the rest of the plan to detect potential future conflicts. The computation of this theoretical resource level takes into account the global borrowing, consumption or production of the resource over past actions, as well as a worst case estimation of the global resource usage of actions currently in execution. Thus IxTeT-eXeC does not yet handle the description of profiles of resource usage with intermediate timepoints during an action. *Because of the worst case estimation, when designing an IxTeT domain model, one should associate the global consumption of a resource during an action with its starting timepoint, the global production with the ending timepoint and the global borrowing with the action duration interval.*

Finally the current implementation restricts *goals to be specified as hold propositions on grounded attributes with fixed values.* This specification still allows to require a specific property to be simply established, or established and maintained during a time interval.

[8] In the LAAS architecture, a specific component, at the execution control level (R2C), also checks that a command will not lead to an unsafe state due to insufficient resources.

2.2 The underlying CSP managers

During the planning process, refinements add constraints to the plan: ordering constraints between timepoints, binding constraints between arguments or values, numeric constraints on resource quantities, etc. These constraints are managed using constraint satisfaction techniques. A *Constraint Satisfaction Problem* (CSP) is commonly defined by a set of variables, their respective domains and a set of constraints between these variables. Finding a solution to this problem amounts to assigning a value to each variable, such that these values are compatible with all the constraints. A CSP is said *consistent* if such a solution exists. In a CSP-based planning process, where least-commitment is desirable, the aim is not to find such a particular solution but to keep all possible solutions by computing the minimal domains for each variable that guarantee the consistency of the CSP.

A planning domain in IxTeT includes two types of variables: temporal variables, or timepoints, ranging over continuous numeric domains, and atemporal variables, ranging either over symbolic or continuous numeric domains. These variables and their associated constraints are handled thanks to two CSP managers: a Time Map and a Variable Map. Mixed constraints between temporal and atemporal variables can also be expressed. They are handled by a supervisor that transfers information from one CSP manager to the other one when required.

2.2.1 Time Map

During the planning process, two types of binary constraints are added to the constraint network by the insertion of new actions and the resolution of conflicts, and handled by the Time Map manager: symbolic ordering constraints[9] (such as the precedence $t_i \prec t_j$) and numeric duration constraints between two timepoints of the form $(t_j - t_i)$ in $[lb_{ij}, ub_{ij}]$. The Time Map manager has to fulfill various types of requests. It is used to incrementally add constraints, to consult the coherence of a specific constraint such as “Is t_1 before t_2?” and to check the global consistency of the constraint network.

The Time Map manager is based on the *Simple Temporal Problem* framework presented in [Dechter 89]. It corresponds to a specific CSP where variables are timepoints and constraints are quantitative constraints of the form $lb_{ij} \leq t_j - t_i \leq ub_{ij}$. A *Simple Temporal Network* (STN) can be represented as a graph where nodes are variables and arcs are labeled by a single numeric interval. Constraint propagation and consistency checking are done using path-consistency algorithms which are polynomial in the number of timepoints n. An STN is conveniently implemented as a $n \times n$ array D where D_{ij} represents for a pair of timepoints (t_i, t_j) the domain of the time distance $(t_j - t_i)$ compatible with all the constraints. The network is inconsistent if $\exists (i,j) / D_{ij} = \emptyset$.

[9] These constraints can also be expressed as numeric binary constraints: $t_i \prec t_j$ is equivalent to $(t_j - t_i)$ in $]0, +\infty[$.

Using this structure, the consultation of the coherence of one constraint is done in constant time: "Is t_1 before t_2?" is equivalent to "$D_{12} \cap]0, +\infty[\neq \emptyset$?", or "Are t_1 and t_2 simultaneous?" is equivalent to "$D_{12} == [0, 0]$?", etc.

Network consistency checking is achieved through a complete propagation of the set of constraints. Global propagation is performed, with a $O(n^3)$ complexity, using the PC-1 algorithm presented in Table 2.1. *Path Consistency* imposes local consistency among triplets of variables until a fixed point is reached or until the network is shown inconsistent. It has been demonstrated in [Dechter 89] that, in the specific case of an STN, one iteration of the PC loop is sufficient and that the resulting network is minimal, i.e. the time intervals D_{ij} represent the minimal domain (with respect to the intersection) compatible with all constraints.

```
**Global-Propagation()**
% Returns true if the network is consistent, false otherwise.
% $D_{ij,i \neq j}$ is initialized with:
%    - $[lb_{ij}, ub_{ij}]$ if a constraint $((t_j - t_i)$ in $[lb_{ij}, ub_{ij}])$ exists,
%    - $]-\infty, +\infty[$ otherwise.
% $D_{ii}$ is initialized with $[0, 0]$.

for k:=0 to $n - 1$ do
  for i,j:=0 to $n - 1$ do
     $D_{ij} \leftarrow (D_{ik} + D_{kj}) \cap D_{ij}$;
     if $D_{ij} = \emptyset$ return false;
return true;
```

Table 2.1: *Global propagation algorithm*

An incremental version of this algorithm, presented in Table 2.3, with a complexity in $O(n^2 + n)$, propagates the constraints between the pair of timepoints (i_0, j_0). It is used when a new constraint $\mathcal{C}_0$ on i_0 and j_0 is added to the network during the planning process (see Table 2.2), and presupposes that the constraints on the other timepoints have already been propagated.

Figure 2.7 displays an example of STN for a simple plan: the rover moves to a location while downloading images. t_0 and t_1 represent respectively the starting and ending timepoints of the plan horizon. t_2 and t_3 are the starting and ending timepoints of the move action, whereas t_4 and t_5 are the starting and ending timepoints of the download-images action. During the planning process, the following constraints are posted: the horizon duration (10min), the actions durations and precedence constraints (represented by dotted lines). The array shows the state of D after a complete propagation.

As said before, both propagation algorithms compute a *minimal network*, in the sense that the D_{ij} intervals correspond to the minimal domains compatible with all constraints.

```
**Add-Constraint**($i_0$,$j_0$,$\mathcal{C}_0$)
% Returns true if the network is consistent, false otherwise.

if ($D_{i_0j_0} \subset \mathcal{C}_0$) do
  return true;
else if ($D_{i_0j_0} \cap \mathcal{C}_0 = \emptyset$) do
  return false;
else do
  $D_{i_0j_0} \leftarrow D_{i_0j_0} \cap \mathcal{C}_0$;
  Incremental-Propagation($i_0$,$j_0$);
  return true;
```

Table 2.2: *Add a constraint $\mathcal{C}_0$ between the timepoints i_0 and j_0*

```
**Incremental-Propagation**($i_0$,$j_0$)
% Assumes that $D$ is completely propagated except for the timepoints $i_0$ and $j_0$.

for i:=0 to $n-1$ do
  $D_{ij_0} \leftarrow (D_{ii_0} + D_{i_0j_0}) \cap D_{ij_0}$;
for i,j:=0 to $n-1$ do
  $D_{ij} \leftarrow (D_{ij_0} + D_{j_0j}) \cap D_{ij}$;
```

Table 2.3: *Incremental propagation algorithm*

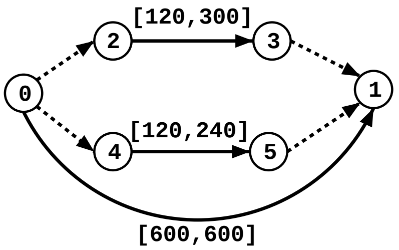
This condition is necessary to guarantee that a complete execution of the plan is possible. The temporal executive decides when to "execute"[10] a timepoint based on the information contained by the STN. The timepoint t_0 representing the origin of the time axis, the possible times at which the event corresponding to a timepoint t_i should occur are given by the domain interval $(t_i - t_0)$ or D_{0i}. If this domain is minimal, for each of its value v_i, there is a global solution to the STP with $(t_i - t_0) = v_i$. The same propagation algorithms are used during execution to update[11] the network and keep it minimal.

2.2.2 Variable Map

Thanks to the extension proposed in [Trinquart 01], the Variable Map manager can handle both symbolic variables ranging other finite domains and numeric variables ranging over

[10] We detail in section 3 what the execution of a timepoint consists in, depending on its type. For an action starting timepoint for instance, it amounts to sending the corresponding launch message to the procedural executive.

[11] When the event corresponding to a timepoint t_i occurs, the timepoint is set to the occurrence time t_{occ} by adding and propagating the constraint $(t_i - t_0) = t_{occ}$.



j i	0	1	2	3	4	5
0	[0,0]	[600,600]	]0,480[	]120,600[	]0,480[	]120,600[
1	[-600,-600]	[0,0]	]-600,-120[	]-480,0[	]-600,-120[	]-480,0[
2	]-480,0[	]120,600[	[0,0]	[120,300]	]-480,480[	]-360,600[
3	]-600,-120[	]0,480[	[-300,-120]	[0,0]	]-600,360[	]-480,480[
4	]-480,0[	]120,600[	]-480,480[	]-360,600[	[0,0]	[120,240]
5	]-600,-120[	]0,480[	]-600,360[	]-480,480[	[-240,-120]	[0,0]

Figure 2.7: *Example of temporal constraint network*

continuous domains. The following constraints apply to both types of variables:

- binding ($?x = ?y$),
- separation ($?x \neq ?y$),
- domain restriction ($?x \in D_x$),
- dependency ($?x \in D_x \Rightarrow ?y \in D_y$).

It is also possible to express numeric constraints of the form:

- $?x + ?y = ?z$,
- $?x - ?y = ?z$,
- $?x * ?y = ?z$,
- $max(?x, ?y) = ?z$,
- $min(?x, ?y) = ?z$,
- $\sum_{i=1..n} ?x_i \leq ?y$ (this constraint is used to bound the sum of consumptions of concurrent temporal propositions on the same resource),
- $\sum_{i=1..n} ?x_i = ?y$ (this constraint is used during execution to update the level of a resource).

During the planning and execution processes, the Variable Map manager needs to fulfill various requests such as checking the coherence of a binding or separation constraint between two variables with the constraint network, incrementally adding and propagating new constraints, or checking the global consistency of the network. Variables are gathered together into equivalence classes, each one representing a set of unified variables which share the same valuation domain.

To compute minimal domains compatible with all constraints, two types of techniques are used:

- Exhaustive propagation techniques guarantee the global consistency of the network. These techniques are expensive and NP-complete.
- Filtering techniques (such as propagation by arc-consistency) guarantee a local consistency. These techniques can be solved in polynomial time but only provide a necessary and not sufficient condition to the network consistency.

During the planning process, a trade-off between both types of technique is done. The constraint network is incrementally built and its consistency is approximated by a filtering technique at each planning step, whereas a complete propagation is performed when a candidate solution plan is available, to check its global consistency.[12]

Thus, each time a new constraint is added, an arc-consistency algorithm performs a local propagation from the equivalence classes concerned by the constraint: the domain of an equivalence class i is restricted until all its values are compatible with the set of constraints involving the variables contained by i. The use of numeric variables and constraints does not especially affect the filtering algorithm and only requires the definition of operators on disjunction of intervals as well as the definition of propagation rules for each numeric constraint.

However, the use of variables ranging over continuous domains does affect the global propagation algorithm. The initial propagation algorithm performed an exhaustive forward checking search and tested every possible instantiations to eliminate values leading to an inconsistent network. The extension proposed by R. Trinquart allows to handle mixed discrete and numeric variables. The exhaustive variable instantiations are replaced by domain splitting guided by the dependency constraints. The principle and algorithms are further detailed in [Trinquart 01, Trinquart 04].

2.2.3 Mixed constraints

A Global Map manager handles a specific type of constraint, mixing both temporal and atemporal variables. Such a constraint has the following form: $?x = c * (t_j - t_i)$ where t_i and t_j are timepoints, $?x$ is a numeric atemporal variable and c is a constant coefficient. It allows the expression of dependency between the effects of an action and its duration. For example, in the rover domain, the duration of a move action is computed by the set of constraints: $?duration = (t_{end} - t_{start})$ with $?distance = ?speed * ?duration$ (and distance depends on the initial and final locations), or the duration of a download-images action

[12] During the execution process, new constraints on atemporal variables are added when a choice of value is made for an action parameter previous to the action launch (restriction of the domain to a singleton) and when the level of a resource is updated. Complete propagation is performed in both cases.

depends on the quantity of memory storage to free and the download rate: $?duration = (t_{end} - t_{start})$ with $?q = ?duration * ?rate$.

The Global Map manager supervises the insertion of a new mixed constraint as two distinct constraints in the corresponding constraint networks. The constraint $t_j - t_i = D_x/c$, with D_x corresponding to the current domain of the variable $?x$, is inserted in the STN, whereas the constraint $?x = c * D_{ij}$, with D_{ij} corresponding to the current time interval between t_i and t_j, is inserted in the atemporal CSP. During the planning process, D_x and D_{ij} may be further restricted. These restrictions are alternatively propagated in both constraint networks, the information on D_x and D_{ij} being transfered by the supervising Global Map manager from one CSP to the other one, and this until a fixed point is reached in both networks.

Likewise, two types of mixed propagation are considered: a "local" one that uses arc-consistency propagation and a "global" one that uses complete propagation for the atemporal network. In both cases, the path consistency algorithm is used for the STN.

However, the use of mixed constraints may have an impact on the temporal network. Indeed, the introduction of a mixed constraint on an atemporal variable ranging over a domain made of a disjunction of intervals induces the splitting of the temporal domain into a disjunction of temporal intervals, thus transforming the STP into a more general TCSP (*Temporal Constraint Satisfaction Problem*, cf. [Dechter 89]).

In the TCSP framework, deciding if a network is consistent is NP-complete and deciding whether it is minimal is NP-hard. Thus path consistency algorithms (such as the PC-1 loop described in Table 2.1 run until a fixed point is reached) are incomplete (may fail to detect an inconsistency). The complexity induced by the disjunctive constraints comes from a possible problem of fragmentation: the total number of intervals in the path consistent network might be exponential in the number of intervals per constraint in the input network. This is due to the relaxation operation used in the PC algorithm: $D_{ij} \leftarrow (D_{ik} + D_{kj}) \cap D_{ij}$, where the number of intervals of $(D_{ik} + D_{kj})$ is bounded by $n_{D_{ik}} * n_{D_{kj}}$, with $n_{D_{xy}}$ representing the number of intervals in the disjunction D_{xy}.

Some approaches have been proposed to handle the complexity of TCSPs. In addition to the path consistency algorithms (such as PC-2 or DPC) which are incomplete, still quite effective to detect inconsistency but present the drawback of possible explosion due to the fragmentation problem, the authors in [Schwalb 97] propose two types of algorithms which are incomplete, somewhat less effective but polynomial. These algorithms are based on approximations. The first one, called ULT (*Upper Lower Tightening*), approximates a disjunction by a single interval made from the lower and upper bounds of the disjunction. The corresponding STN is propagated and its consistency is checked. Finally, the initial TCSP is intersected with the resulting STN. The second algorithm, called LPC (*Loose Path Consistency*), is more efficient than ULT. It is similar to a PC algorithm whose intersection operator has been redefined as a *loose intersection* operator[13]. These algorithms can be

[13] Let $T = \{I_1, I_2, I_3, \ldots\}$ and $S = \{J_1, J_2, J_3, \ldots\}$ be two disjunctions of intervals, the loose intersection

used as filtering or preprocessing functions to detect inconsistencies, but do not compute a minimal network.

However, it is interesting to note that, practically, the PC algorithm often converges to the minimal network ([Dechter 89]). Other experimental results [Shapiro 99] also show that the fragmentation problem is a worst-case scenario hard to achieve. The number of intervals in the output of a sum of two interval sets ($D_{ik} + D_{kj}$) effectively varies as the product $n_{D_{ik}} * n_{D_{kj}}$ if n_D is limited (< 10), but quickly decreases when $n_D > 15$.

Keeping a minimal network is a necessary condition for the execution process which relies on the minimal domains to decide the consistent execution time of timepoints. A minimal network can be computed thanks to a PC filtering algorithm followed by an expensive backtrack search algorithm if the resulting network is not minimal (see [Dechter 89]). This method is not completely satisfying in the context of temporal execution with frequent updates and constraint propagation in a limited amount of time.

Thus the issue of TCSP management still needs to be addressed. In the meantime, we restrict the expression of mixed constraints to the ones involving atemporal variables ranging over a single continuous interval and which will not contain gaps in the course of propagation. This restriction does not affect the model of the rover domain. In the constraint $?duration = (t_{end} - t_{start})$ with $?distance = ?speed * ?duration$, for example, $?speed$ represents the speed range of the rover from 0.03 to 0.1 $m.s^{-1}$ ($?speed \in [0.03, 0.1]$) and $?distance$, originally ranging over $]-\infty, +\infty[$, is computed depending on the numeric values of $?x_1$, $?x_2$, $?y_1$ and $?y_2$ following the rigid property represented in Figure 2.2. Thus, the temporal constraint network remains a Simple Temporal Problem.

2.2.4 Discussion with respect to plan execution

More realistic models

The use of CSPs allows the design of a more realistic model of the world. As said before, mixed constraints are useful to represent interactions between the effects of an action and its duration. Besides, most of the time, an action does not last an exact duration nor use an exact amount of a resource. The use of numeric variables with domains ranging over continuous intervals allows to reflect part of the uncertainty on quantitative effects and durations on to the model of actions.

between T and S corresponds to the disjunction $\{I'_1, I'_2, I'_3, \ldots\}$ with $I'_i = [lb_i, ub_i]$, lb_i and ub_i being the lower and upper bounds of the intersection $I_i \cap S$. The loose intersection is equivalent to the intersection in the case of STN.

Flexibility

The plans produced are not instantiated, linearized and time-stamped set of actions. Instead the plan is flexible, keeping for each temporal and atemporal variable a minimal domain which only reflects the necessary constraints. As discussed in section 1.2.2, such order-constrained plans with unbound variables are worthwhile, the plan being further constrained during the plan execution according to the actual timings and events.

The STN framework also provides an efficient support to the plan execution. The temporal executive relies on the plan's temporal network to determine what to do and when. The temporal data (occurrence time of events) are regularly updated during execution by the simple insertion and propagation of a domain restriction constraint. Furthermore, we take advantage in the IXTET-EXEC framework of the temporal flexibility to try and repair the plan after an execution failure and eventually postpone the execution of some actions, while executing the valid part of the plan.

2.3 Partial Order Causal Link Planning with unbound variables

Planning consists in determining, from a knowledge of an initial situation and the set of possible actions, the course of actions to achieve a set of goals. In classical planning, two main strategies have been applied. Planning can be viewed as a search in the state space, i.e. the search space is a graph where a node represents a state of the world and a transition from a state S_1 to a state S_2 corresponds to an action that is applicable in S_1 and whose effects lead to S_2. Planning then amounts to finding a path from the initial state to a final state where all goals are satisfied. The other strategy performs a search in the plan space. A rough and incomplete plan is progressively expanded and refined by the insertion of actions and constraints until a complete and valid plan is found. A node in the search tree then corresponds to a partial plan and a transition from P_1 to P_2 to a plan transformation that fixes a specific flaw of P_1.

The plan space approach produces plans whose actions are partially ordered. A widely used strategy to keep track of the interactions between the actions consists in recording them through a set of causal links that represent and protect the establishment of a property by an action. The IxTeT planning system follows such a *Partial Order Causal Link* (POCL) planning approach. Using the commonly adopted principle of least commitment during the search: to postpone as late as possible a choice that is not mandatory now, IxTeT will produce plans that are also partially instantiated. Moreover, the classical POCL framework has been extended to handle time and scheduling problems with resource allocations.

In this section, we briefly present the search space and the planning algorithm in the IxTeT framework, we then detail the possible flaws and plan transformations, and finally discuss the heuristics used to guide the search.

2.3.1 Search in the plan space

A partial plan in the classical POCL planning framework

In the POCL planning framework, the search process explores a tree in the partial plan space. A *partial plan* is generally defined as a 4-tuple (A, C, L, F) where:

- A is a set of partially instantiated actions,
- C is a set of constraints on temporal and atemporal variables appearing in A and L,
- L is a set of causal links,
- and F is a set of flaws.

A *causal link* $a_i \xrightarrow{p} a_j$ denotes a commitment by the planner that a proposition p of action a_j is established by an effect of action a_i. The precedence constraint $a_i \prec a_j$ and binding constraints for variables of a_i and a_j appearing in p are in C. The set F may contain two types of flaw: *open conditions* and *threats*. Open conditions correspond to propositions that have not yet been established in the plan, by the initial situation or by the actions in A, and protected by a causal link. A threat appears in the plan if an action a_k has effects in contradiction with the establishment of the proposition p by the causal link $a_i \xrightarrow{p} a_j$ and $(a_i \prec a_k \prec a_j)$ is consistent. The execution of a partial plan containing such flaws can lead to a situation where an action is not applicable.

Using this definition, the planning problem, i.e. the initial state and the goals, is expressed as a specific partial plan $\mathcal{P}_I$ containing two types of "null" actions: one "initial" action whose effects correspond to the description of the initial state, and several "goal" actions, each one having a top-level goal as a precondition. The set of flaws F of $\mathcal{P}_I$ contains the preconditions of the goal actions. $\mathcal{P}_I$ is the root node of the search tree. A planning step involves selecting a flaw in the partial plan and resolving it, resulting in a new partial plan. This process is repeated until a solution plan is found.

A partial plan $\mathcal{P}$ is a solution plan to the planning problem if it forms a *complete* and *consistent* plan. $\mathcal{P}$ is **consistent** if the corresponding temporal and atemporal constraint networks are consistent. These networks record for each variable its minimal valuation domain. Thus a partial plan stands for a set of plan interpretations, where an **interpretation** is a linearization of the partial plan by a complete and consistent assignment of values to the temporal and atemporal variables. $\mathcal{P}$ is **complete** if each of its possible interpretations forms a path in the state space between the initial state and a state in which all goals are satisfied. It is equivalent to characterize a solution plan as a partial plan where the set of constraints C is consistent and the set of flaws F is empty. This last characterization can serve as termination condition for the search process.

A partial plan in IxTeT

In IxTeT, the POCL framework has been extended to the chronicle approach (presented in section 2.1.4). The search process relies on a chronicle on a set of logical attributes and a set of resource attributes:

$$\mathcal{C} = (Evt_{\mathcal{C}}, Hold_{\mathcal{C}}, Use_{\mathcal{C}}, Prod_{\mathcal{C}}, Cons_{\mathcal{C}}, C_{\mathcal{C}})$$

This structure is similar to the partial plan (A, C, L, F) defined above. Each action in A, as well as the planning problem corresponding to the "initial" and "goal" null actions, is expressed as a chronicle on subsets of logical and resource attributes. $Evt_{\mathcal{C}}$, $Hold_{\mathcal{C}}$, $Use_{\mathcal{C}}$, $Prod_{\mathcal{C}}$ and $Cons_{\mathcal{C}}$ gather the temporal propositions present in the set of actions A. The causal links are materialized as specific *hold* propositions (cf. the following definition of an established proposition). Thus L is contained by a subset of $Hold_{\mathcal{C}}$. The temporal, atemporal and mixed constraints on the variables appearing in A and L are present in $C_{\mathcal{C}}$. Finally, the set of flaws F is also contained by $Evt_{\mathcal{C}}$, $Hold_{\mathcal{C}}$, $Use_{\mathcal{C}}$, $Prod_{\mathcal{C}}$ and $Cons_{\mathcal{C}}$. A chronicle $\mathcal{C}$ may involve three types of flaws: **open conditions**, **threats** and **resource**

conflicts. The two first ones are similar to the classical flaws presented above, whereas the latter is specific to the management of scheduling problems.

In short, **open conditions** are *event* and *hold* propositions that have not yet been established (the value of the logical attribute is not justified by the initial scenario or an effect of a previous action). The establishment of a proposition is recorded and protected thanks to causal links (materialized by specific *hold* propositions). A grounded logical attribute can take only one value at a time. During the planning process, causal links can be "threatened" by another overlapping proposition. Thus **threats** correspond to pairs of *event* and *hold* propositions on the same logical attribute that potentially have the same arguments, potentially overlap and whose values can be in conflict. Finally **resource conflicts** are sets of temporal propositions on the same resource type that potentially overlap and over-consume an allocated resource.

Let define more formally the flaws in the IxTeT framework. A temporal proposition on a logical attribute is either

$$hold(AttName(?x_1, \dots, ?x_n) :?v_1, (t_1, t_2))$$
$$\text{or } event(AttName(?x_1, \dots, ?x_n) : (?v_1, ?v_2), t_1).$$

It can eventually be labeled as **explained** or **contingent**. Explained propositions are used to assert the initial state of the world, whereas contingent propositions are used to describe the evolution of contingent attributes on which the planning agent has no means of control. In both cases, the planning system considers that the propositions are already established[14] and does not try to justify them.

Otherwise, a temporal proposition on a logical attribute in a partial plan is said **established** if the two following conditions hold.

Definition 2.3.1 (established proposition). *A temporal proposition is* ***established*** *iff:*
1. there is an ***establisher event*** *in the plan:* $event(AttName(?x'_1, \dots, ?x'_n) : (?v'_1, ?v'_2), t'_1)$ *such that* $?v'_2 =?v_1$, $\forall i\ ?x_i =?x'_i$ *and* $t'_1 \leq t_1$ *hold; and*
2. if $(t'_1 < t_1)$, *there is a* ***causal link*** *that maintains the value of the attribute between the establisher and the established proposition. This causal link is represented by a proposition:* $hold(AttName(?x''_1, \dots, ?x''_n) :?v''_1, (t''_1, t''_2))$ *with the constraints* $?v''_1 =?v_1$, $\forall i\ ?x''_i =?x_i$, $t''_1 = t'_1$ *and* $t''_2 = t_1$.

This definition leads to the one of an open condition flaw as:

Definition 2.3.2 (open condition). *Is an open condition any temporal proposition on a logical attribute that is not established in the current partial plan.*

Besides, a chronicle may contain two types of threats (cf. Figure 2.8):

[14] Moreover, these establishments do not need to be protected by causal links. The planner can neither introduce actions that have effects occurring at the beginning of the plan, nor actions that have some effect on a contingent attribute.

Definition 2.3.3 (threat). *A **threat** is a possibly conflicting pair of temporal propositions in the current partial plan, i.e. either:*

1. a pair (e,cl) where

$$\begin{aligned} e = event(AttName(?x'_1, \ldots, ?x'_n) : (?v'_1, ?v'_2), t'_1) \text{ and} \\ cl = hold(AttName(?x_1, \ldots, ?x_n) :?v_1, (t_1, t_2)) \end{aligned}$$

such that the current partial plan does not contain one of the constraints $\{(t'_1 < t_1); (t'_1 = t_1$ *and* $?v_1 =?v'_2); (t_2 < t'_1); (t_2 = t'_1$ *and* $?v_1 =?v'_1); (?x_1 \neq ?x'_1); \ldots; (?x_n \neq ?x'_n)\}$*; or*

2. a pair (h,cl) where

$$\begin{aligned} h = hold(AttName(?x'_1, \ldots, ?x'_n) :?v'_1, (t'_1, t'_2)) \text{ and} \\ cl = hold(AttName(?x_1, \ldots, ?x_n) :?v_1, (t_1, t_2)) \end{aligned}$$

such that the current partial plan does not contain one of the constraints $\{(t'_2 < t_1); (t_2 < t'_1); (?x_1 \neq ?x'_1); \ldots; (?x_n \neq ?x'_n); (?x_1 =?x'_1$ *and* $\ldots$ *and* $?x_n =?x'_n$ *and* $?v_1 =?v'_1)\}$.

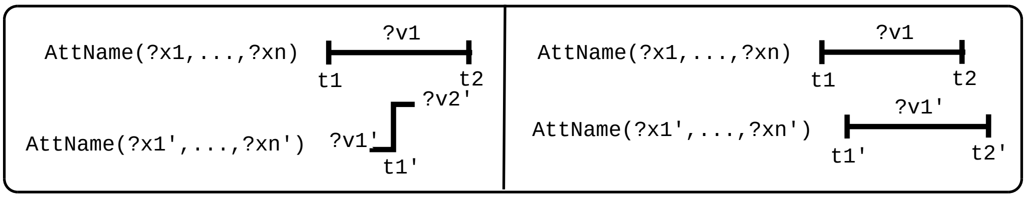


Figure 2.8: *Threats*

To introduce the third type of flaw, namely resource conflicts, we first define a criterion of coherence of a partial plan with respect to resources. Let consider a chronicle $\mathcal{C}$ and one of its interpretations $\mathcal{C}_i$, corresponding to a complete assignment of values to the timepoints and the atemporal variables (including the variables representing the borrowed, consumed and produced amounts of resources) compatible with the constraints in $C_{\mathcal{C}}$.

We note $C(r)$ the capacity of the allocated resource $R(r)$. Let note respectively $Prod_i(r)$, $Cons_i(r)$ and $Use_i(r)$ the sets of *produce*, *consume* and *use* propositions on $R(r)$ present in the interpretation $\mathcal{C}_i$. A production p in $Prod_i(r)$ of a quantity q_p occurs at a timepoint t^p. A consumption c in $Cons_i(r)$ of a quantity q_c occurs at a timepoint t^c. Finally, a borrowing u in $Use_i(r)$ of a quantity q_u occurs between the timepoints st^u and et^u. Using these notations, the variation at a timepoint t of the available quantity of an allocated resource $R(r)$ is equal to:

$$\Delta(R(r), t) = \sum_{\substack{u \in Use_i(r)/ \\ et^u = t}} q_u + \sum_{\substack{p \in Prod_i(r)/ \\ t^p = t}} q_p - \sum_{\substack{u \in Use_i(r)/ \\ st^u = t}} q_u - \sum_{\substack{c \in Cons_i(r)/ \\ t^c = t}} q_c$$

An interpretation is said coherent with respect to resources if, for each allocated resource, its available quantity $Q(R(r))$ remains positive over time. That is, $\mathcal{C}_i$ is coherent with respect to resources iff, for each allocated resource and at each timepoint t:

$$Q(R(r), t) = C(r) + \sum_{t'/t' \leq t} \Delta(R(r), t') \geq 0.$$

Finally, a chronicle is necessarily coherent with respect to resources if all its possible interpretations are coherent with respect to resources.

From this global criterion, a local one, relying on the notion of *minimal critical sets* (MCS), has been defined to check the coherence w.r.t. resources of a chronicle. To introduce the notion of MCS, we will refer to the first implementation of resource management in IxTeT, as described in [Laborie 95a, Laborie 95b]. Thus, we restrict the problem by considering only borrowings, consumptions and productions of fixed quantities on non-parameterized types of resource. An extension of the notion of MCS to the case of variable amounts and resource types with different possible allocations is presented in section 2.4.

First, the representation of resource usage is unified to simplify the algorithms: the *produce* and *consume* propositions are translated into equivalent *use* propositions. Indeed, the consumption of a quantity q at time t can also be viewed as a borrowing of the same quantity between t and the end of the horizon. Thus:

$$\mathit{consume}(R() : q, t) \Leftrightarrow \mathit{use}(R() : q, (t, +\infty))$$

And the production of a quantity q at time t can be viewed as a production of this quantity at the early beginning, quantity that is not available until time t. Or, similarly, the production corresponds to an increment of the resource capacity C_R by q along with a borrowing of this quantity between the origin and t. Thus:

$$\mathit{produce}(R() : q, t) \Leftrightarrow \begin{cases} \mathit{use}(R() : q, (0, t)) \\ C_R = C_R + q \end{cases}$$

The plan now contains a set of partially ordered *use* propositions. A *critical set* corresponds to a potential contention of a resource, i.e. a set of potentially overlapping *use* propositions whose cumulative usage exceeds the resource capacity. We can then define a resource conflict as:

Definition 2.3.4 (resource conflict). *A* ***resource conflict*** *is a set of* use *propositions* $\{u_1, \ldots, u_k\}$ *present in the current partial plan* $\mathcal{P}$ *such that, if* $u_i = \mathit{use}(R() : q_i, (st_i, et_i))$*:*

1. $\mathcal{P}$ *does not contain one of the constraints* $\{(et_i < st_j)_{i \neq j}\}$*; and*
2. $\sum_{i=1}^{k} q_i > C_R$.

A *minimal critical set* S is then a resource conflict, such that any subset of S is not a resource conflict. And, as proved in [Laborie 95a], a chronicle (or partial plan) is necessarily coherent w.r.t. resources iff it does not contain any minimal critical set.

In conclusion, a chronicle $\mathcal{C}$ represents a solution plan iff it does not contain any flaw, i.e.:

- the constraint networks are consistent,
- each proposition on a logical attribute is established,
- the chronicle does not contain any conflicting pair $(event, hold)$ or $(hold, hold)$,
- the chronicle does not contain any minimal critical set.

2.3.2 IxTeT planning process

As introduced before, a POCL planner explores a tree in the plan space: a node is a partial plan and a branch corresponds to a resolver that fixes a flaw. We detail in the next subsection what are the possible resolvers for each type of flaw. Briefly, they correspond to the insertion of new constraints (promotion, demotion, separation, etc.) that solve conflicts or to the insertion of a new action to establish a proposition or produce a resource.

```
Plan-one-step($\mathcal{C}$)

% 1. Analysis:
% find the flaws contained by $\mathcal{C}$ and compute a disjunction of resolvers for each flaw
$(flaws, resolvers) \leftarrow analyse(OpenCond(\mathcal{C}) \cup Threats(\mathcal{C}) \cup MCS(\mathcal{C}))$;
if ($flaws = \emptyset$ and $C_{\mathcal{C}}$ is consistent) then
   return($\mathcal{C}$);  % solution plan
% 2. Flaw selection
select one flaw $f \in flaws$;
if ($resolvers(f) = \emptyset$) then
   return(failure);  % backtrack
% 3. Resolver choice
non deterministically choose a resolver $r$ in $resolvers(f)$;
% 4. Resolver insertion
$\mathcal{C}' \leftarrow$ insert($\mathcal{C}$,$r$);
if ($C_{\mathcal{C}'}$ is not consistent) then
   return(failure);  % backtrack
```

Table 2.4: *Planning step*

Table 2.4 presents the principle of a planning step. It is composed of four phases: first the partial plan is analyzed to find its flaws[15] and to propose a disjunction of potential resolvers for each flaw, a flaw is then selected and one of its possible resolvers is chosen and

[15] In IxTeT however, the plan is not completely analyzed at each step. As presented later, only a subset of the attributes is considered during the analysis and an agenda of the flaws is maintained and incrementally updated at each step.

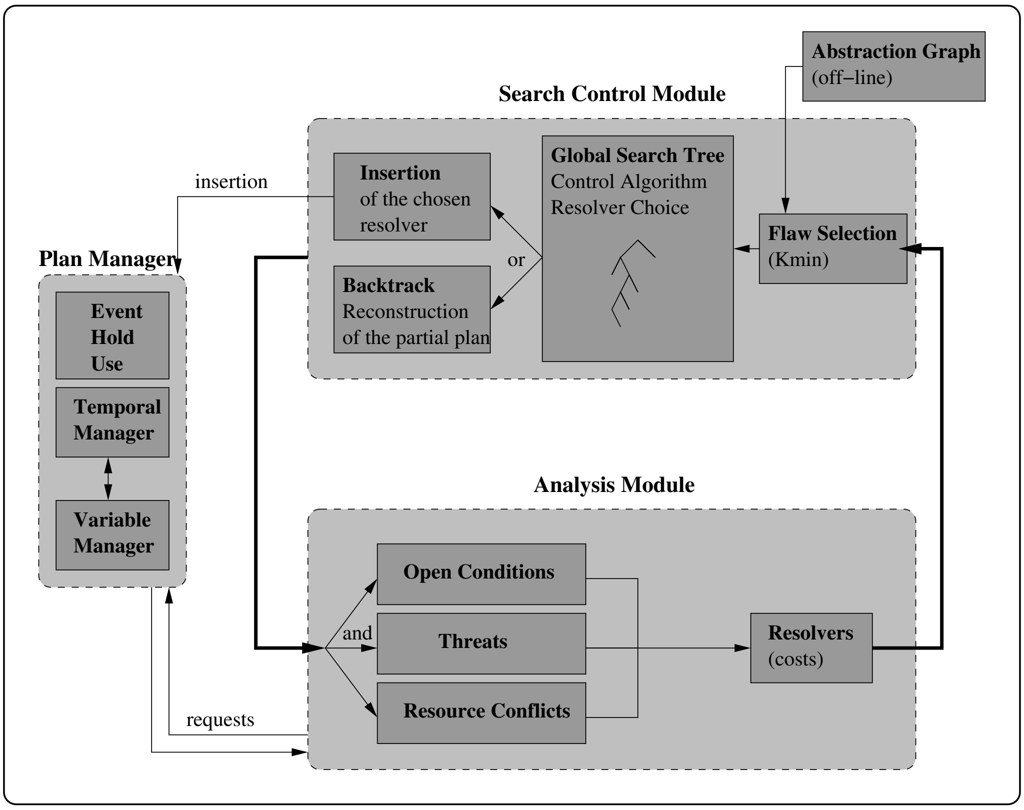


Figure 2.9: *IxTeT modules*

inserted, resulting in a new partial plan. This planning step is repeated until a solution plan is found.

Two sorts of decisions are made during a planning step: the selection of a flaw, and the non deterministic choice of a resolver for that flaw. In the same time, two failures may arise: the plan contains a necessary flaw, or a resolver insertion leads to inconsistent constraint networks. To guarantee the completeness of the search, the planning algorithm backtracks on the resolver choice points: a previous node in the tree is chosen and another resolver, different from the one that led to the failure, is applied. It is not necessary to call the flaw selection into question, since all flaws in the plan need to be addressed anyway. However, a smart choice on the order in which the flaws are addressed can significantly reduce the search space. This planning procedure is sound and complete: if a solution to the planning problem exists, the process terminates and returns a solution plan.

Figure 2.9 presents the organization in modules of the IxTeT planner. The Analysis module is in charge of detecting the flaws of the current partial plan. It computes for each flaw the complete set of its resolvers and associates with each resolver a *cost* that is used to guide the future non deterministic choice. Besides, flaws are ranked according to an abstraction hierarchy computed off-line and satisfying the *Ordered Monotonicity Property*

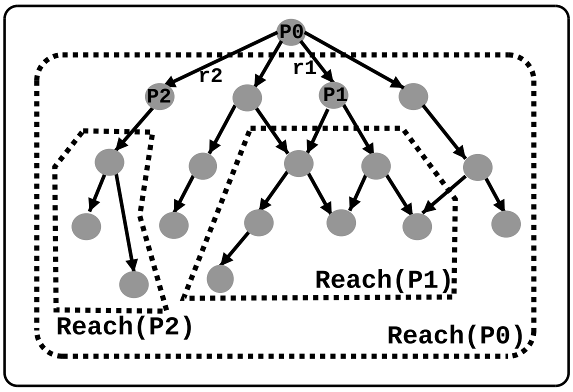


Figure 2.10: *Reachability in the plan space*

[Knoblock 94]. The analysis encompasses only flaws that belong to the level of abstraction that is currently considered by the search process. Unlike the classical use of hierarchies, this abstraction level is dynamically built during the search process.

Among the set of flaws of a same abstraction level, the Search Control module selects a flaw, apart from its type, according to an *opportunistic least-commitment* strategy, based on an estimation of the commitment induced by each resolver (its "cost"). Moreover, the search tree is controlled either by a near-admissible A_ϵ algorithm or by an ordered depth first search strategy[16]. Finally, the insertion and propagation through the constraint networks of a resolver are achieved by the Plan Manager module. We detail the Analysis module and the Search Control module in the two following subsections.

2.3.3 Analysis module

The Analysis module detects flaws in the partial plan, computes the set of resolvers for each flaw and associates a cost with each resolver. These costs are used to guide the flaw selection and resolver choice.

The flaw selection strategy proposed in [Ghallab 94, Laborie 95a] extends the principle of least commitment to temporal planning. A least-commitment strategy defines a selection criterion applicable to all types of flaw. The flaw preferably chosen at a search step is the one for which the branching factor in the search tree is minimal, i.e. the flaw with fewest resolvers. This principle has been extended in IxTeT to estimate the commitment of a resolver as its influence on the reduction of the solution plans reachable from the current partial plan in the plan space. And a flaw is selected according to its number of resolvers but also according to the easiness to choose between its resolvers the least committed one.

Considering P a node in the partial plan space, we note *Reach*(P) the set of partial

[16] The plan repair process and replanning process in IxTeT-eXeC use the ordered depth first search strategy.

plans that are reachable from P (cf. Figure 2.10). We also note $Reach_i(P)$ the set of all possible interpretations[17] of the plans in $Reach(P)$. If P_0 represents the initial problem (the root node) and $Sol(P_0)$, $Sol_i(P_0)$ represent respectively the set of solution plans to this problem and the set of the interpretations of these solution plans, then $Sol_i(P_0)$ is a subset of $Reach_i(P_0)$.

The insertion of a resolver r_1 to solve a flaw of P_0 leads to the partial plan $P_1 = P_0 \oplus r_1$. The set of reachable plan interpretations is reduced and the solution plan interpretations contained by $Reach_i(P_0) \backslash Reach_i(P_1)$ are removed.

The notion of least commitment then corresponds to the choice, among a set of possible resolvers, of the one that removes fewest possible solutions. Under the assumption that the solution plan interpretations are uniformly distributed over $Reach_i(P)$, a criterion consists in preferring the resolver that reduces the least the set of reachable plan interpretations.

Thus, if it is possible to measure the size μ of the set of reachable plan interpretations, the commitment of a resolver r from a partial plan P is estimated by:

$$commit(P, r) = 1 - \frac{\mu(Reach_i(P \oplus r))}{\mu(Reach_i(P))}$$

The value of $commit(P, r)$ ranges from 0 (the resolver is redundant) to 1 (the resolver leads to an inconsistent plan).

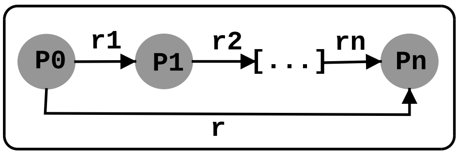


Figure 2.11: *Conjunction of elementary resolvers*

If a resolver r is a conjunction of elementary resolvers $(r_1, \dots, r_n)$ (see Figure 2.11) with

$$\begin{aligned} commit(P_{j-1}, r_j) &= 1 - \frac{\mu(Reach_i(P_{j-1} \oplus r_j))}{\mu(Reach_i(P_{j-1}))} = 1 - \frac{\mu(Reach_i(P_j))}{\mu(Reach_i(P_{j-1}))} \text{ for } j = 1..n, \text{ and} \\ commit(P_0, r) &= 1 - \frac{\mu(Reach_i(P_n))}{\mu(Reach_i(P_0))}, \end{aligned}$$

then

$$1 - commit(P_0, r) = \prod_{j=1}^{n} (1 - commit(P_{j-1}, r_j)).$$

If two elementary resolvers r_i and r_j are of a different type, it is reasonable to assume that the ratio of instances deleted by the insertion of r_i is the same if r_i is posted in P or in

[17] cf. page 58.

$P \oplus r_j$. This assumption leads to the formula:

$$1 - commit(P_0, r) = \prod_{j=1}^{n} (1 - commit(P_0, r_j)),$$

and, since $commit(P_x, r_x) < 1$, a first order approximation of the commitment of the resolver r applied to P_0 is:

$$commit(P_0, r) \approx \sum_{j=1}^{n} commit(P_0, r_j).$$

The measurement function μ is difficult to establish, especially in the case of the insertion of a new action. So, the commitment of r is approximated by a cost function associated with each type of resolver. These cost functions are intuitive estimations of the ratio of instantiations of P removed by the insertion of a resolver.

In the following, we detail for each type of flaw how it is detected, what are the possible resolvers and what are the associated cost estimations.

Threats

Threat Protection	event	established event	hold	established hold
hold	N	N	N	Y
established hold	Y	Y	Y	Y

Figure 2.12: *Different categories of threats*

Detection - The threat detection requires to search through the pairs $(e, h) \in Event_{AttName} \times Hold_{AttName}$ and $(h_1, h_2) \in Hold^2_{AttName}$ for each logical attribute $AttName$ of the current abstraction level. In IxTeT, the establishment status of a temporal proposition is updated during the search. Thus the potential threats can be classified according to the establishment status of the involved propositions, as shown in Figure 2.12. The flaw analysis module can be parameterized to not take into account some of the threat categories. The threats involving two established propositions are necessarily considered to guarantee the completeness of the search process. On the other hand, it can be worth ignoring threats on non established propositions, since their future establishment will correspond to an extension of the temporal interval on which the value needs to be protected.

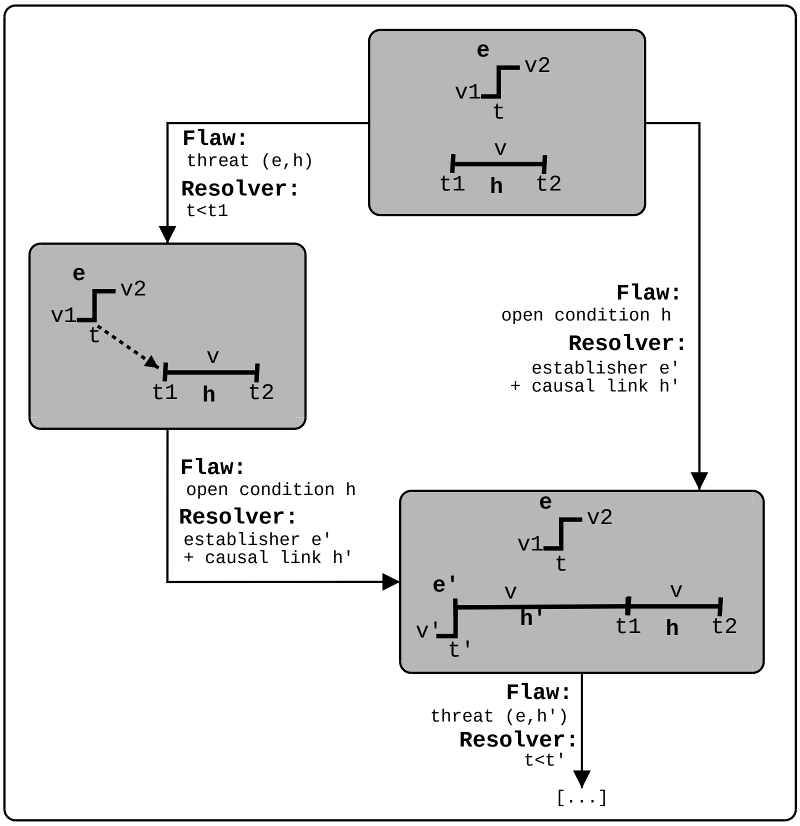


Figure 2.13: *Threat on non established propositions*

Figure 2.13 illustrates such a scenario, where it appears useless to solve the threat (e,h). Finally, Figure 2.12 presents the default control parameters used in IxTeT (Y/N: the threat is taken into account/ignored).

Resolvers - The possible resolvers for a threat are the insertion in the current partial plan of temporal and atemporal constraints leading to the promotion or demotion of the threatening proposition, or the separation of the attribute arguments. More precisely, considering a threat (e, h) where

$$\begin{aligned} e &= \mathit{event}(\mathit{AttName}(?x'_1, \ldots, ?x'_n) : (?v'_1, ?v'_2), t'_1) \text{ and} \\ h &= \mathit{hold}(\mathit{AttName}(?x_1, \ldots, ?x_n) : ?v_1, (t_1, t_2)) \end{aligned}$$

the disjunction of resolvers is made of the constraints $\{[(t'_1 < t_1)], [(t'_1 = t_1), (?v_1 = ?v'_2)], [(t_2 < t'_1)], [(t_2 = t'_1), (?v_1 = ?v'_1)], [(?x_1 \neq ?x'_1)], \ldots, [(?x_n \neq ?x'_n)]\}$ that are compatible with the constraint networks. And, considering a threat (h_1, h_2) where

$$\begin{aligned} h_1 &= \mathit{hold}(\mathit{AttName}(?x'_1, \ldots, ?x'_n) : ?v'_1, (t'_1, t'_2)) \text{ and} \\ h_2 &= \mathit{hold}(\mathit{AttName}(?x_1, \ldots, ?x_n) : ?v_1, (t_1, t_2)) \end{aligned}$$

the disjunction of resolvers is made of the constraints $\{[(t'_2 < t_1)], [(t_2 < t'_1)], [(?x_1 \neq ?x'_1)], \ldots, [(?x_n \neq ?x'_n)], [(?x_1 = ?x'_1), \ldots, (?x_n = ?x'_n), (?v_1 = ?v'_1)]\}$ that are compatible with the constraint networks.

Costs - An ordering resolver (promotion/demotion) corresponds to the insertion of a constraint $(t_1 < t_2)$, or similarly $(t_2 - t_1)$ in $]0, +\infty[$. Posting this constraint results in the deletion of the negative instances of the time interval $D_{t_1t_2} = [lb_{t1t2}, ub_{t1t2}]$ in the STN[18]. Thus the commitment of an ordering resolver is approximated by:

$$cost(P, (t_1 < t_2)) = \frac{-min(lb_{t_1t_2}, 0)}{ub_{t_1t_2} - lb_{t_1t_2}} * c_T$$

A separation resolver corresponds to the insertion of a constraint $?x \neq ?y$. If we note D_x and D_y the valuation domains of the variables $?x$ and $?y$ before the constraint is posted, then there are $card(D_x) * card(D_y)$ possible instantiations of the pair $(?x, ?y)$. The constraint forbids all instantiations such that $?x = ?y$, i.e. $card(D_x \cap D_y)$ instantiations. The commitment of a separation resolver is then computed using the formula:

$$cost(P, (?x \neq ?y)) = \frac{card(D_x \cap D_y)}{card(D_x) * card(D_y)} * c_S$$

Other resolvers imply a binding constraint $?x = ?y$. Posting this constraint leads to the reduction of the domains of $?x$ and $?y$ to the values in $D_x \cap D_y$. The cost of this constraint is then computed using the formula:

$$cost(P, (?x = ?y)) = \frac{card(D_x) * card(D_y) - card(D_x \cap D_y)}{card(D_x) * card(D_y)} * c_B$$

Finally, the cost of a conjunction of such constraints $C = (c_1 \vee \ldots \vee c_n)$ is defined as the sum of the elementary costs: $cost(P, C) = \sum_{i=1}^{n} cost(P, c_i)$.

The constant coefficients c_T, c_S and c_B can be tuned by the user (their default value is 1), in order to prioritize some type of resolver.

Open conditions

Detection - In IxTeT, the establishment status of a proposition is explicitly represented, and so the detection of open conditions is straightforward.

[18] If the temporal network is a TCSP, the cost is computed using the formula

$$cost(P, (t_1 < t_2)) = \frac{length(D'_{t_1t_2})}{length(D_{t_1t_2})} * c_T$$

where $D'_{t_1t_2} = D_{t_1t_2} \cap] - \infty, 0]$ and $length([a_1, b_1] \vee \ldots \vee [a_n, b_n]) = (b_1 - a_1) + \ldots + (b_n - a_n)$, the intervals $[a_i, b_i]$ being disjoint.

Resolvers - If $h = hold(AttName(?x_1, \ldots, ?x_n) :?v, (t, *))$ (similarly $e = event(AttName (?x_1, \ldots, ?x_n) : (?v, *), t))$ is an open condition, then fixing such a flaw consists in finding an establisher event $e_{est} = event(AttName(?x'_1, \ldots, ?x'_n) : (*, ?v_{est}), t_{est})$ either in the current partial plan or thanks to the insertion of a new action A, and protecting the establishment by a causal link $cl = hold(AttName(?x_1, \ldots, ?x_n) :?v, (t_{est}, t))$.

With each potential establisher event is associated one of the following conjunctions of elementary resolvers: $[(t_{est} < t) \wedge (?x'_1 =?x_1) \wedge \ldots \wedge (?x'_n =?x_n) \wedge (?v_{est} =?v) \wedge cl]$ for an intern establisher, or $[A, (t_{est} < t) \wedge (?x'_1 =?x_1) \wedge \ldots \wedge (?x'_n =?x_n) \wedge (?v_{est} =?v) \wedge cl]$ for an extern establisher. Finally, the disjunction of resolvers for an open condition gathers the conjunctions of all possible establishments.

Costs - The cost of an intern establishment is computed using the formula:

$$\begin{aligned} cost(P, causal\ link) &= cost(P, (t_{est} < t)) \\ &\quad + cost(P, (?v_{est} =?v)) + \sum_{i=1}^{n} cost(P, (?x'_i =?x_i)) \\ &\quad + coef_{cl} * DurationRatio \end{aligned}$$

$DurationRatio$ reflects the fact that a causal link forbids events on the grounded attribute to occur during a part of the plan between the establisher event and the established proposition. Thus the least committed causal link is the one that minimizes the constraint $DurationRatio = \frac{lb_{t_{est}t}}{lb_{horizon}}$ where $lb_{t_{est}t}$ is the minimal time distance between t_{est} and t, and $lb_{horizon}$ is the minimal plan duration. $coef_{cl}$ is a constant coefficient. This control parameter is tuned by the user.

The cost of an extern establishment is computed using the formula:

$$cost(P, action\ insertion) = C_A$$

C_A corresponds to an estimation of the cost of the action insertion. Two strategies have been adopted. The first one allows the user to specify a fixed cost for each action model. The second one performs a look ahead to estimate the amount of new open conditions induced by the action insertion. As presented in [Ghallab 94], this procedure relies on a subgoal decomposition and expands an AND/OR tree where the nodes AND represent the set of new subgoals introduced by an action and the nodes OR represent the set of actions that can possibly establish a given subgoal. The expansion of the tree is limited to a fixed depth d and controlled by an AO^* algorithm. The cost of an action is computed as the sum of the costs of its subgoals, whereas the cost of a subgoal is computed as the minimal cost of its children actions or of establisher events present in the current partial plan. This strategy presents two weaknesses. First, it does not take into account the interactions between actions and resources. Second, the expansion of the tree at each open condition analysis is expensive. Practically, best performances are obtained for $d \in \{0, 1, 2\}$ (cf. [Laborie 95a]). By default, the IxTeT planner uses the strategy of fixed costs.

Resource conflicts

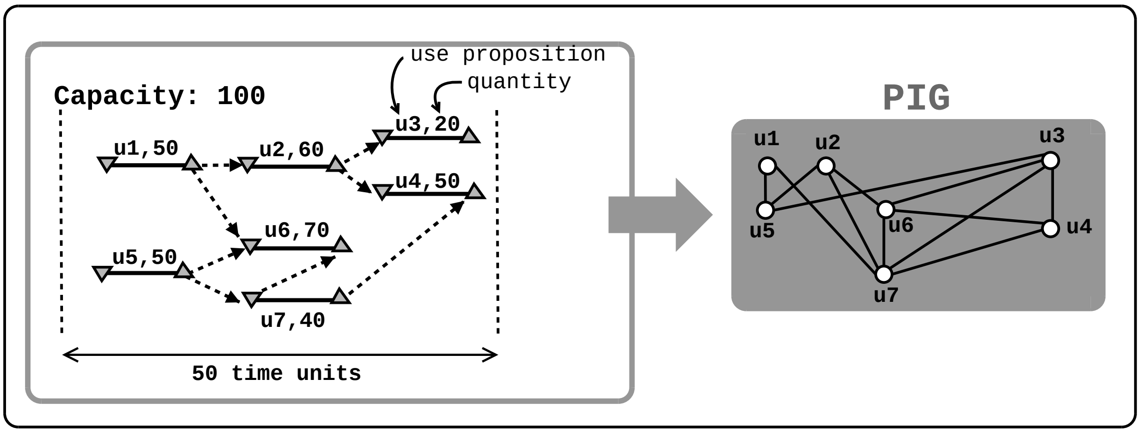


Figure 2.14: *Example of PIG*

Detection - Once again, we present the case involving non-parameterized resources and usages of fixed quantities as developed in [Laborie 95a, Laborie 95b]. For each resource attribute R with a maximal capacity C_R, the analysis module looks for the *Minimal Critical Sets* (MCS) present in the current partial plan.

An MCS is a set of *use* propositions $U =< u_1, .., u_k >$, with $u_i = use(R() : q_i, (st_i, et_i))$, such that:

- all propositions u_i potentially overlap,
- $\sum_U q_i > C_R$,
- $\forall U' \subset U, \sum_{U'} q_i \leq C_R$.

This search is limited to the detection of the smallest MCS which are the most interesting ones in the context of a least commitment approach, since they are solved by fewer resolvers.

The MCS detection is based on a specific structure: a *Possible Intersection Graph* (PIG). This non-oriented graph represents the temporal relations between the *use* propositions on a given resource attribute R. A node of the PIG stands for a *use* proposition, and there is an edge between two nodes i and j if their corresponding propositions u_i and u_j overlap in some temporal interpretation of the partial plan. Figure 2.14 shows an example of PIG for the set $< u_1, ..., u_7 >$, where each proposition lasts exactly 10 time units, all propositions occur in a time interval of 50 time units and are constrained by the precedence constraints represented by the dotted arrows[19].

An MCS is equivalent to a minimal over-consuming clique in the PIG. As shown in [Laborie 95a], the PIG has the advantage of being a weakly triangulated graph, for which

[19] Another precedence constraint between the end of the proposition u_5 and the beginning of the proposition u_4 is induced by the propagation of the numeric constraints.

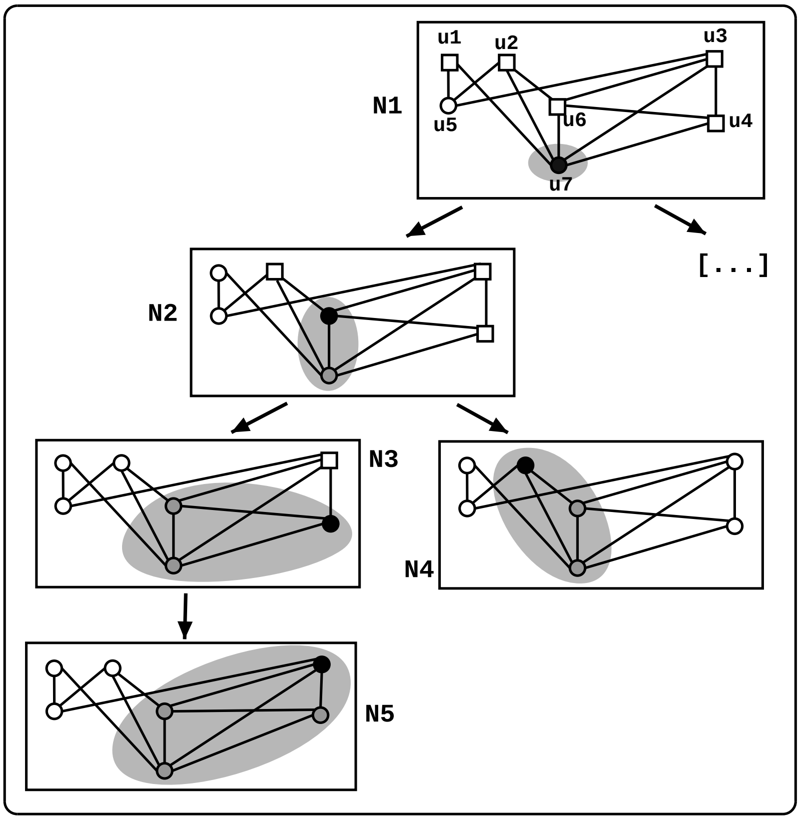


Figure 2.15: *Principle of the MCS search tree*

the maximum clique problem for instance can be solved in polynomial time (whereas it is an NP-complete problem for an arbitrary graph). An exhaustive search of all MCS is intractable, thus the analysis process develops a specific search tree to find the smallest over-consuming cliques.

A node N in this search tree represents a clique $\Gamma(N)$, and the search principle consists in scanning cliques whose size increases when exploring a branch of the tree. If $(N_0, \ldots, N_k)$ is a path from the root to a leaf, then $\Gamma(0) = \emptyset \subset \Gamma(N_1) \subset \ldots \subset \Gamma(N_k)$.

More formally, a node N is defined by the triplet $< \Delta(N), P(N), \Phi(N) >$. $\Delta(N)$ is the set of propositions that contribute to the enlargement of the current clique $\Gamma(N)$. $P(N)$ is the pool of candidate propositions to enlarge the current clique in the children nodes. $\Phi(N)$ contains the sub-cliques of $\Gamma(N)$ which are MCS. Figure 2.15 illustrates the principle of this search tree: at each node N the shadowed ellipse, black points and squares represent respectively the current clique $\Gamma(N)$, $\Delta(N)$ and $P(N)$.

A node N is developed in three steps. First, $P(N)$ is partitioned into disjoint sets $\Delta(M)$, where each node M corresponds to a son of node N. Second, the set $P(M)$ is computed for each node M. Finally, $\Phi(M)$ is computed by searching the MCS which have

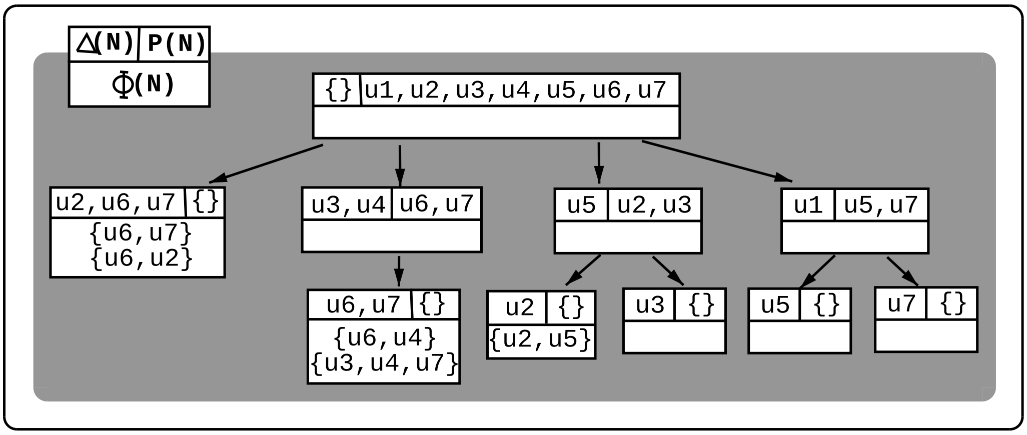


Figure 2.16: *Example of search tree developed with the DECLIC algorithm*

at least one proposition in each $\Delta(N_f)$, $N_f \in fathers(M) \cup M$. To limit the breadth and depth of the search tree, $P(N)$ is partitioned into sets whose cardinality can be greater than 1. The algorithm DECLIC detailed in [Laborie 95a] computes simultaneously $\{\Delta(M)\}_{M \in Sons(N)}$ and $\{P(M)\}_{M \in Sons(N)}$ through a vertex ordering of $P(N)$ similar to the computation of a perfect combination for a triangulated graph. Figure 2.16 details the search tree developed by the DECLIC algorithm for the example given in Figure 2.14.

The most expensive step is the enumeration of the sub-cliques of $\Gamma(N)$ to find the ones corresponding to MCS. For a node N, the method computes all cliques with one proposition in each $\Delta(N_f)$, $N_f \in fathers(N) \cup N$. These cliques are then progressively enlarged by adding one proposition from the sets $\{\Delta(N_f)\}_{N_f \in fathers(N) \cup N}$. A clique γ is added to $\Phi(N)$ if $\sum_\gamma q > C_R$ and $\sum_\gamma q - min_\gamma(q) \leq C_R$ (the conflict is minimal). This enumeration method is in $O(2^k)$ where k is bounded by the size of the maximum clique of the PIG[20]. The search tree is explored using a breadth first search to detect first the smallest MCS (the size of a conflict in $\Phi(N)$ is greater than or equal to the depth of the node).

Resolvers - Given an MCS $U =< u_1, \ldots, u_n >$ with $u_i = use(R() : q_i, (st_i, et_i))$, it can be solved either by posting one of the following precedence constraints $\{(et_i \prec st_j)_{(i,j) \in [1..n] \times [1..n], i \neq j}\}$ or by inserting an action that produces the resource.

The MCS can be solved by at most $n(n-1)$ precedence constraints. However, some of them are redundant. A minimization procedure computes for an MCS U the minimal disjunction of temporal resolvers by removing the resolvers r that would imply another resolver r' of U. This minimization allows to prune some over-constraining branches of the plan search tree. If we consider the MCS $< u_3, u_4, u_7 >$ in the previous example, it

[20] However, this maximum clique can be quite big. If N actions in the partial plan consume the same resource, the corresponding *use* propositions form a clique of size N. In IxTeT-eXeC, as time progresses and actions are executed, the overlapping resource propositions are aggregated in one equivalent proposition, thus drastically reducing the analysis search space.

can be solved by $\{(et_3 \prec st_4), (et_3 \prec st_7), (et_7 \prec st_3), (et_7 \prec st_4), (et_4 \prec st_3)\}$, but, since $r = (et_4 \prec st_3)$ implies $(et_7 \prec st_3)$, r will not be considered.

The MCS U over-consumes the quantity $OC = \sum_{u_i \in U} q_i - C_R$. Each action model A_p that produces a quantity greater than or equal to OC of the resource R at a timepoint t_p can be inserted in the plan before one of the timepoints $\{st_i\}_{i=1..n}$ to solve the conflict. A similar minimization procedure is applied to the set of possible resolvers involving A_p: $\{(A_p, t_p \prec st_i)\}_{i=1..n}$. If the constraint $(st_a \prec st_b)_{a \neq b}$ exists, then $(t_p \prec t_a)$ implies $(t_p \prec t_b)$ and $(A_p, t_p \prec t_a)$ will not be considered.

Costs - A resolver is either an ordering constraint or an action insertion. The corresponding costs are the same as those described before.

2.3.4 Search Control module

We detail in this subsection the heuristics used to guide the search. We first consider how the flaws of a partial plan are ranked according to an abstraction hierarchy. We then describe the opportunistic least-commitment approach used to select a flaw and the strategy of search tree exploration. We finish with a discussion on the performances of the planner.

Abstraction hierarchy

The use of abstraction hierarchies is a widely adopted strategy to explore large search trees. In the context of non-linear planning, the principle consists in sorting flaws and gathering them into classes, each class being associated with a specific abstraction level. The control of the search then relies on this classification: the planner fixes all flaws of a given level before taking into account the flaws of a less abstract level. This classification is also used to improve the performances of the flaw analysis by limiting it to the flaws belonging to the current abstraction level.

Diverse abstraction hierarchies have been proposed. The one exploited in IxTeT verifies the *Ordered Monotonicity Property* (OMP) defined in [Knoblock 94]. If P_i is an abstract plan, i.e. a solution plan that does not contain any flaw of the abstraction level i, then P_{i-1} is a refinement of the abstract plan P_i if P_{i-1} is an abstract plan itself and differs from P_i just by the addition of resolvers for the flaws of the abstraction level $(i-1)$. The OMP states that *for all abstract plans, every refinement of those plans leaves the literals established in the abstract plan unchanged.* Such a hierarchy can reduce the sources of backtrack: *the cause for backtracking arises not because of an interaction across abstraction levels, but because in some cases no refinement exists* (necessary flaws and inconsistent constraints). Furthermore, an ordered monotonicity hierarchy can be automatically generated from the description of the domain.

The contribution of the approach proposed in [Garcia 98, Garcia 95] and implemented in IXTET is twofold: the classical STRIPS-like abstraction formalism has been adapted to the chronicle representation, and a dynamic hierarchy has been defined. Indeed, unlike previous works (ABSTRIPS [Sacerdoti 74], ABTWEAK [Yang 90], ALPINE [Knoblock 94]), where static hierarchies are totally ordered off-line and exploited as-is during the search, this approach automatically generates a set of admissible hierarchies off-line. The effective abstraction hierarchy is then dynamically managed during planning following a least-commitment strategy.

Since a flaw in IXTET always affects a unique attribute name, it has been chosen to characterize the abstraction level of a flaw by its attribute name.

Hypothesis 2.3.1 (one level per attribute). *Abstraction levels are sets of flaws. Two flaws with the same attribute name are necessarily in the same level.*

The hierarchy between abstraction levels is then required to satisfy the following property:

Property 2.3.1 (Ordered Monotonicity Property for IXTET). *For all possible current partial plans, every potential transformation of those plans in order to solve flaws only creates new flaws belonging to the current or next abstraction levels.*

The automatic generation of such an abstraction hierarchy relies on the extraction of sufficient conditions on the abstraction levels (i.e. on sets of attribute names) from the syntactic description of the domain actions. These conditions amount to the definition of a partial order on the set of attribute names. We note att_p the attribute name present in the temporal proposition p (either *hold*, *event*, *use*, *produce* or *consume*).

Definition 2.3.5 (Ordering constraints $\prec$ on attributes). *For every action A of the domain, if m is either an* event *proposition on a controllable attribute, or a* produce *proposition on a resource attribute of A, and if p is any temporal proposition of A, then $att_p \prec att_m$.*

Based on these relations between attributes, an order between sets of attributes that are $\prec$-equivalent is defined as:

Definition 2.3.6 (Ordering constraints $<$ on classes of $\prec$-equivalent attributes). *For each attribute name att, we note $\overline{att} = \{att'/(att \prec att') \wedge (att' \prec att)\}$ its class of $\prec$-equivalence. Then $\overline{att} < \overline{att'}$ iff $\overline{att} \neq \overline{att'}$ and $att \prec att'$.*

Finally, an acyclic abstraction graph, generated automatically from the description of actions, represents the partial order $<$ on the classes of $\prec$-equivalent attributes. Its nodes correspond to the $\prec$-equivalence classes, and its oriented edges represent the $<$ ordering constraints. During the planning process, a constraint $\overline{att} < \overline{att'}$ implies that it should not be allowed to solve a flaw on an attribute in $\overline{att}$ until all flaws concerning attributes in $\overline{att'}$ have been fixed.

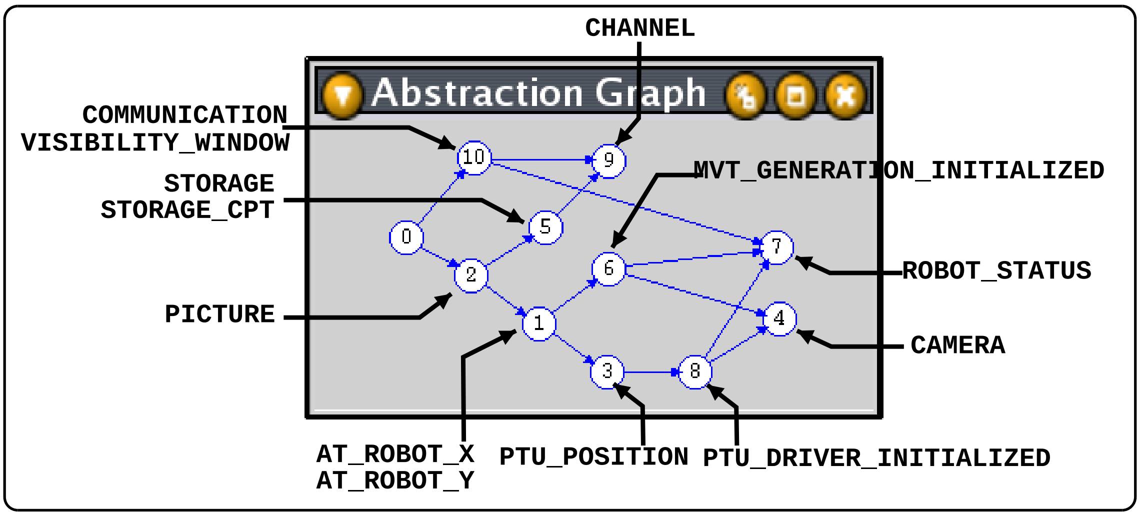


Figure 2.17: *Abstraction graph for the rover domain*

We slightly modified the conditions on the abstraction levels defined in [Garcia 95] to differentiate contingent from controllable attributes. Indeed, contingent conditions impose constraints on the plan and it is preferable to address flaws on contingent attributes as soon as possible in the search process. A contingent attribute is automatically detected as an attribute that appears exclusively in *hold* temporal propositions in the description of actions (the agent has no means to change its value). The following redefinition of the $\prec$ relation allows to place a contingent attribute as high as possible in the abstraction graph without calling the satisfaction of the OMP into question.

Definition 2.3.7 (Ordering constraints $\prec$ on contingent and controllable attributes). *For every action A of the domain, if m is an* event *proposition on a controllable attribute, or a* produce *proposition on a resource attribute, or an* hold *proposition on a contingent attribute of A, and if p is any temporal proposition of A, then $att_p \prec att_m$.*

Besides, a reservoir resource attribute R is internally associated with a complementary resource attribute R_CPT which has a mirror behavior (cf. subsection 2.1.3). R and R_CPT are $\prec$-equivalent. Figure 2.17 shows the abstraction graph for the rover domain, where $VISIBILITY\_WINDOW$ is a contingent attribute, and $STORAGE\_CPT$ is a complementary resource.

Whereas the abstraction graph is computed off-line, the effective hierarchy is defined on-line, according to the evolution of the search. An abstraction state is associated with each node of the planning search tree:

Definition 2.3.8 (abstraction state). *An abstraction state is a pair (S, C) where S is the set of attributes that have been completely solved so far, and C is the set of attributes that are currently in process.*

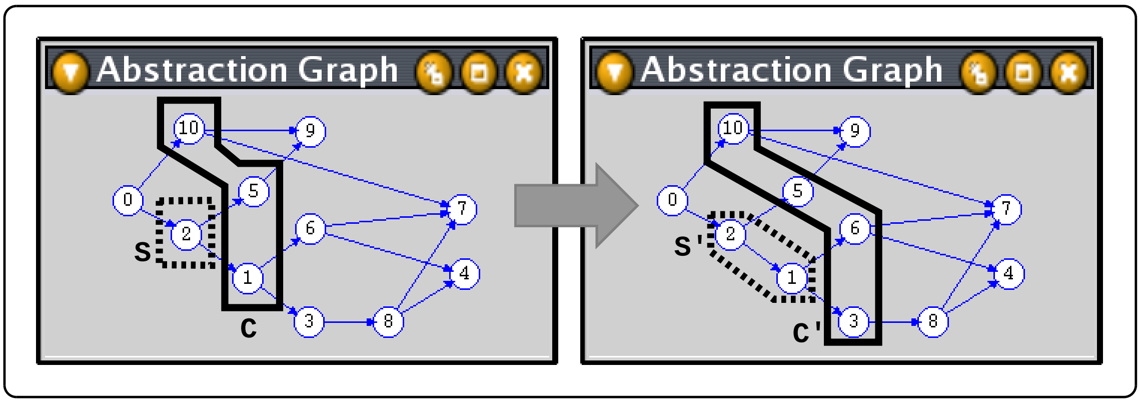


Figure 2.18: *Example of consecutive abstraction states*

Let denote the attribute name involved in the flaw f by att_f. Hence, the current abstraction level L can be formally defined as

$$L = \{f/att_f \in C\}$$

and the definition of a hierarchy is equivalent to the specification of the update rules for the abstraction state during the planning process. A classical hierarchical approach would consist in waiting for the set C to become empty before computing a new set C' and a new level L'. In the IxTeT approach however, new attributes can be immediately added to C if the $\prec$ constraints are respected. The initial abstraction state associated with the root node is set to (S_0, C_0) with $S_0 = \emptyset$ and $C_0 = \{att/\{att'/\overline{att} < \overline{att'}\} = \emptyset\}$, and (S, C) is progressively updated according to the following rule:

Definition 2.3.9 (abstraction state update). *When a set $\overline{att_{solved}}$ of $\prec$-equivalent attributes is completely solved, $\overline{att_{solved}}$ is removed from C, added to S, and C is recomputed from the new set of solved attributes S' as:*

$$\begin{aligned} S' &= S \cup \overline{att_{solved}} \\ C' &= (C \backslash \overline{att_{solved}}) \cup \{att_{new}/(\overline{att_{new}} < \overline{att_{solved}}) \wedge (\{att/\overline{att_{new}} < \overline{att}\} \subset S')\} \end{aligned}$$

Figure 2.18 shows two consecutive abstraction states. The hierarchy defined this way is an ordered monotonic one. A proof can be found in [Laborie 95a]. Basically, the treatment of a flaw $f \in L$ will not introduce any flaw involving an attribute in S. Indeed, if a new flaw f' appears with the insertion of a new action in order to solve f, then, according to the definition of $\prec$ constraints: $\overline{att_{f'}} < \overline{att_f}$ or $\overline{att_{f'}} = \overline{att_f}$.

Apart from its role in the ordering of flaws, the abstraction graph provides other interesting features. Notably, it records which attributes are linked to the primary effects of each action (we call them **main attributes**). Primary effects specify the real purpose of the actions, as opposed to the secondary effects which are only its side effects. During the planning process, an operator is selected to establish an open condition or to produce a resource based on its primary effects. In IxTeT-eXeC, the information about the main attributes of actions is also used during the execution process, when the plan is partially invalidated, to determine which causal links should be broken to allow the insertion of specific new actions.

Flaw selection

The Search Control module has to select a flaw among the ones that affect attributes in the set C of the current abstraction state. A flaw selection criterion, based on the resolver cost functions, has been defined. These cost functions are used to determine the least committed resolver for each flaw (i.e. with the minimal cost), and the *opportunity* $Opp(\varphi)$ of solving a flaw φ corresponds to the estimation of the easiness to make a choice between its resolvers. The isolation degree of the optimal resolver is quantified by the formula:

$$K = \frac{1}{Opp(\varphi)} = \sum_{r \in resolvers(\varphi)} \frac{1}{1 + cost(r) - cost_{min}}$$

The Search Control module selects the flaw with the minimal factor K. If all resolvers have the same cost, K is equal to the number of resolvers, and if a flaw is determinist, the factor K is minimal and equal to 1.

Search tree control

Given a flaw, the choice of a resolver amounts to decide which branch of the search tree the planner should develop. The user can choose between two algorithms to control the tree exploration: either an A_ϵ algorithm or an Ordered Depth First (ODF) search strategy. Both algorithms rely on the heuristic information $f = g + h$ that estimates, for each node N, the cost (g) of the path explored between the root and N combined with the cost (h) of the path to be explored between N and a solution plan.

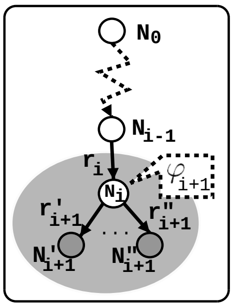


Figure 2.19: *Development of node N_i*

At each step of the search, a pending node N_i is chosen (cf. Figure 2.19) and developed by: inserting its associated resolver (r_i), analyzing the resulting partial plan, selecting a flaw (φ_{i+1}) and creating a successor node for each of its possible resolvers ($N'_{i+1}, N''_{i+1}, \ldots$). Each successor node is characterized by its corresponding resolver and a heuristic estimate f, and added to the set of pending nodes.

The ODF search strategy develops next the successor node with the minimal f. If a node has no successor, it backtracks to the last branching node and develops its next best successor. The A_ϵ algorithm also prioritizes a depth-first exploration and develops next the best successor node if it is acceptable, i.e. if its estimate f satisfies: $f \leq (1 + \epsilon) f_{min}$, where f_{min} is the smallest estimate in the set of pending nodes. If there is no acceptable successor, the algorithm backtracks to the pending node with the minimal f.

For a node N_i, $g(N_i)$ corresponds to the cumulative commitment of the plan transformations along the path from N_0 to N_i:

$$g(N_i) = \sum_{k=1}^{i} cost(r_k).$$

$h(N_i)$ corresponds to an estimation of the distance between the partial plan P_i and a solution plan, hence it is computed as the sum of the minimal costs induced by the resolution of the flaws present in P_i:

$$h(N_i) = \sum_{\varphi \in flaws(P_i)} min_{r \in resolvers(\varphi)}(cost(r))$$

The main difference between the two search control algorithms is that, in the case of the ODF search, the estimate f is only used to compare resolvers (and choose the one with the minimal cost), whereas, in the A_ϵ algorithm, the function f can also serve as a cause for backtrack, the aim being to combine a depth first search with the search of a good quality plan (i.e. a plan with as little actions as possible and the least committed constraints).

However the estimate f defined above is not always efficient to guide the choice of the best pending node on which to backtrack in an A_ϵ algorithm. Indeed, the function h entails several approximations:

- It does not take into account the interactions between flaws and the fact that a same resolver can fix several flaws.
- It does not take into account the possibility for a resolver to introduce new flaws (especially if the cost of an action insertion is computed using the fixed costs strategy).
- Estimating the function h for a node requires to insert the corresponding resolver and analyze the flaws present in the resulting partial plan. Since the information $h(N_{i+1})$ is needed at the end of the development of its parent node N_i, and since it is very expensive to completely expand and analyze all successor nodes of N_i, h is further approximated by:

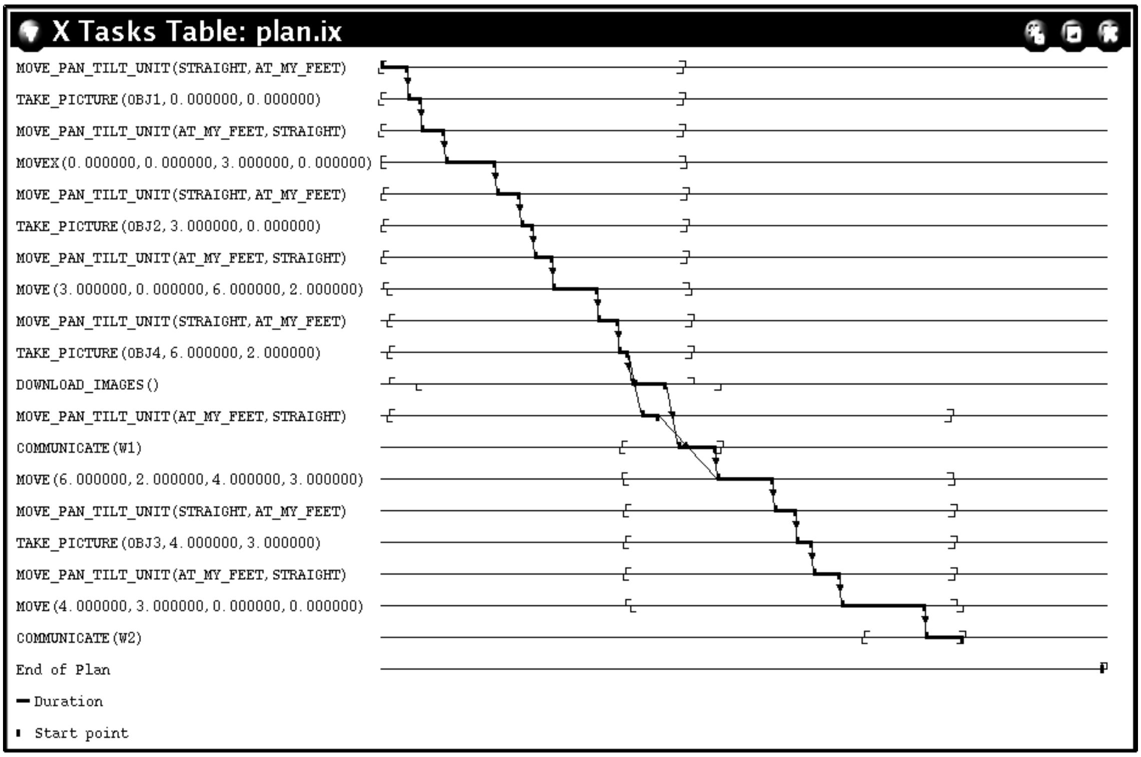
  $$\tilde{h}(N_{i+1}) = h(N_i) - min_{r \in resolvers(\varphi_{i+1})}(cost(r))$$

  This formula is coherent with the previous approximations: the partial plan P_{i+1} is assumed to correspond to P_i with exactly one flaw solved (φ_{i+1}) and no additional flaws. Hence, at each node N_i, the function f is computed for each successor node as:

  $$f(N_{i+1}) = g(N_i) + cost(r_{i+1}) + \tilde{h}(N_{i+1})$$

- Finally, this function h is even less informed since $h(N_i)$ is computed based on a partial analysis of P_i that considers only flaws belonging to the current abstraction level. On one hand, it is expensive to perform a complete analysis at each node and the abstraction ordering of flaws considerably reduces the number of backtracks, but on the other hand, the comparison of estimates f for nodes at different abstraction levels does not provide enough information to decide on the best backtrack node. The search process is likely to backtrack on nodes at a high-level of abstraction, since the estimation of their distance to a solution plan does not encompass the lower level flaws.

Example



X Tasks Table: plan.ix

```
MOVE_PAN_TILT_UNIT(STRAIGHT,AT_MY_FEET)
TAKE_PICTURE(OBJ1,0.000000,0.000000)
MOVE_PAN_TILT_UNIT(AT_MY_FEET,STRAIGHT)
MOVEX(0.000000,0.000000,3.000000,0.000000)
MOVE_PAN_TILT_UNIT(STRAIGHT,AT_MY_FEET)
TAKE_PICTURE(OBJ2,3.000000,0.000000)
MOVE_PAN_TILT_UNIT(AT_MY_FEET,STRAIGHT)
MOVE(3.000000,0.000000,6.000000,2.000000)
MOVE_PAN_TILT_UNIT(STRAIGHT,AT_MY_FEET)
TAKE_PICTURE(OBJ4,6.000000,2.000000)
DOWNLOAD_IMAGES()
MOVE_PAN_TILT_UNIT(AT_MY_FEET,STRAIGHT)
COMMUNICATE(W1)
MOVE(6.000000,2.000000,4.000000,3.000000)
MOVE_PAN_TILT_UNIT(STRAIGHT,AT_MY_FEET)
TAKE_PICTURE(OBJ3,4.000000,3.000000)
MOVE_PAN_TILT_UNIT(AT_MY_FEET,STRAIGHT)
MOVE(4.000000,3.000000,0.000000,0.000000)
COMMUNICATE(W2)
End of Plan
```

Figure 2.20: *Example of plan for the rover domain*

Figure 2.20 shows an example of plan produced by IxTeT for the rover domain. The initial scenario contains 7 goals: one goal requires the rover to be back in position (0,0) at the end of the mission, 2 goals require the rover to communicate with the ground station during a visibility window, 4 goals correspond to images taken at four different locations. The square brackets represent the temporal flexibility, the arrows, the precedence constraints between the timepoints of the actions.

Table 2.5 shows how the performances of the planner evolve when the number of "image" goals in the above problem increases. These tests have been run on a Pentium IV, using the ODF strategy.[21]

Nb of "image" goals	Nb of developed nodes	Nb of backtrack points	Depth of solution node	Time (s)
1 image	49	3	46	0.5
2 images	61	4	57	1
3 images	91	6	85	3.7
4 images	134	9	121	31.5
5 images	167	12	149	60.9
6 images	>1000	>192	/	>1286.5

Table 2.5: *Examples of planning performances*

Conclusion and improvement

We briefly summarize the influence of the diverse guidance criteria on the global performances of the search process.

- The use of a dynamic abstraction hierarchy essentially reduces the causes for backtrack and speeds up the search at each node by limiting the analysis space.
- In practice, the use of the estimate f, based on the approximation of the resolver costs, is quite efficient to discriminate the best resolvers (especially from a set of resolvers of the same type) and generally produces good quality plans.
- However, this heuristic f is not sufficiently informed to efficiently guide the backtrack process. Thus, we have implemented the ODF strategy which relies on chronological backtracking. The search control in IxTeT-eXeC is based on it.

The backtrack process presents another difficulty. Once the backtrack node has been chosen, diverse strategies can be deployed to recover the data of the corresponding partial plan. A space-consuming one consists in recording each partial plan. The one implemented in IxTeT rebuilds the backtrack partial plan by successively inserting and propagating in the initial plan the resolvers associated with the parents of the backtrack node. However this strategy is not satisfactory in the context of plan repair in IxTeT-eXeC. The worst-case duration of the execution cycle depends, among other things, on the worst case duration of a planning step. In practice, this worst case often corresponds to the reconstruction of a deep backtrack node (in the order of a few seconds in our experiments).

In order to bound the time spent in the reconstruction of the partial plan of a specific node, we have modified the backtrack strategy. The partial plan is cached every r developed nodes. The plan of a node N is built by recovering the plan corresponding to its closest parent node that has been cached and successively inserting in this plan the

[21] The A_ϵ algorithm could not solve these problems in a reasonable amount of time.

resolvers leading to N. This caching frequency r is specified by the user. If $r = 0$, the plan is built starting from the initial plan; if $r = 1$, all the plans are cached; if $r = 5$, plans are cached every 5 steps; etc.

Figure 2.21 shows the influence of plan caching, for the rover domain, on the total planning duration and on the memory size used at the end of the planning process. These tests have been made on five of the planning problems presented in Table 2.5. It results from this sample that recording each plan is both space and time consuming, but that recording every 5 nodes for instance, can considerably improve the performances (the initial planning duration for the “5 images” problem was 60.9s and has been reduced to 38.9s). Recording the partial plan every 10 nodes seems to be a good trade-off between time and space for the application we are concerned with.

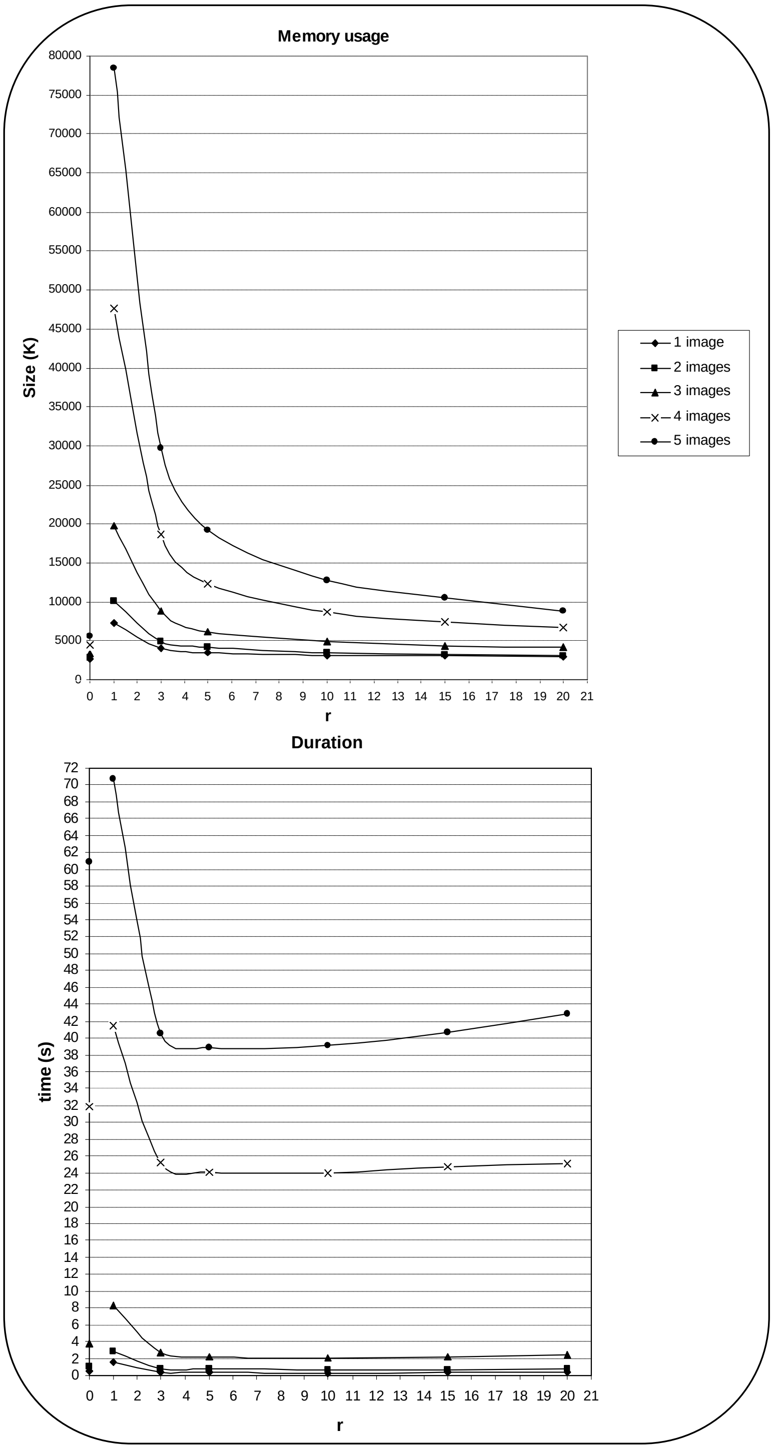


Figure 2.21: *Variations of memory usage and planning durations for r={0,1,3,5,10,15,20}*

2.3.5 Discussion with respect to plan execution

Adaptability of the search in the plan space

The main drawback of non-linear planners is a lack of powerful heuristics to guide the search. Such a planner can usually produce plans from scratch limited to 20-30 actions.

The search in the plan space however provides a good framework to perform plan adaptations. The IxTeT planner has already been used to accomplish incremental planning [Gout 99] and plan merging operations in a context of multi-robot cooperation [Gaborit 96]. In the first case, the temporal planner IxTeT has been integrated in the decisional level of a software architecture designed for an autonomous observation satellite. In this context, it has been used as an incremental planner that could handle successive goal requests under certain restrictions: the insertion of a new goal and the actions that achieve it did not interfere with the global plan previously found and currently executed. In the second case, P. Gaborit defined operators on temporal plans in order to distribute the generation of a global plan between several agents. These operators include the union of plans and the insertion of new goals inside a plan. They rely on a disruption of chains of causal links, that allows to re-establish propositions thanks to the insertion of other actions.

The work pursued in IxTeT-eXeC focuses more on the adaptation of a plan in reaction to failures, state updates or new goals. The integration of these unexpected events may break causal links or reveal resource conflicts, and the plan repair process tries to re-establish these lost properties by performing a search in the plan space, starting from a partially invalidated plan. This plan contains the flaws revealed by the report integration, but also actions that remain flawless and executable. It is further invalidated by the relaxation of causal constraints (using the technique proposed by P. Gaborit), but only on specific attributes (selected using the information on main attributes of actions given by the abstraction graph) to limit the number of planning decisions and keep as many executable actions as possible.

2.4 Resource management extension

The resource management presented so far is not totally adequate to describe the resource usages involved by more realistic domains. In order to handle conditions such as "The energy consumed at the end of a navigation task varies with the time needed to cover the distance to the destination", "The memory space consumed by an image varies with the compression rate", or "An image can be stored indifferently on disk1 or disk2", we need to extend the resource representation in two directions:

- a resource type R has several potential allocations (represented by a symbolic variable $?r$),
- the quantities borrowed, consumed or produced during an action can be expressed as numeric variables $?q$.

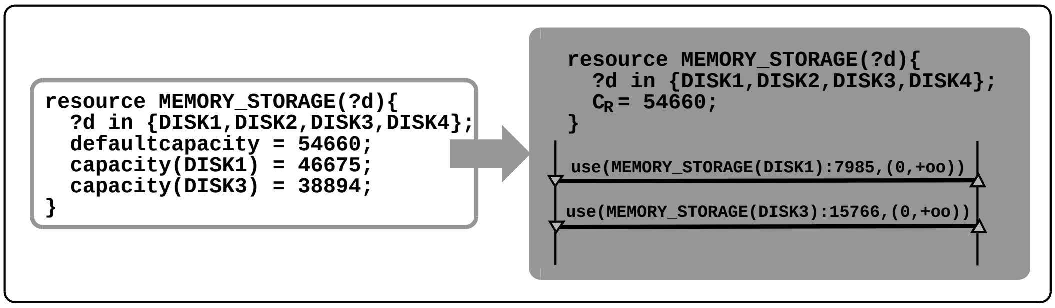
These variables can be linked to other variables to express resource usages depending on the parameters of an action or on its duration, etc. Such modifications also improve the flexibility of the final plan and make it more robust to the execution hazards.

In this section, we consider how this extension of the expressiveness and flexibility impacts the resource management, i.e. the definition of a resource conflict, its detection, and its possible resolvers.

2.4.1 Resource conflict

We consider now a resource type R with several potential allocations $D_R = \{r_1, \ldots, r_n\}$, each having a specific initial capacity $C_0(r_i)$.

The internal representation and algorithms in IxTeT consider that all resource allocations share the same initial capacity C_R, associated with the resource type, and defined as the maximal initial specific capacity: $C_R = max_{r_i \in D_R}(C_0(r_i))$. For each resource allocation r_i such that $C_0(r_i) < C_R$, a borrowing proposition of the quantity $C_R - C_0(r_i)$ over the plan horizon is added to the plan. Figure 2.22 illustrates the declaration of a resource and its corresponding internal representation.



```
resource MEMORY_STORAGE(?d){
  ?d in {DISK1,DISK2,DISK3,DISK4};
  defaultcapacity = 54660;
  capacity(DISK1) = 46675;
  capacity(DISK3) = 38894;
}
```

```
resource MEMORY_STORAGE(?d){
  ?d in {DISK1,DISK2,DISK3,DISK4};
  CR = 54660;
}
```

use(MEMORY_STORAGE(DISK1):7985,(0,+oo))
use(MEMORY_STORAGE(DISK3):15766,(0,+oo))

Figure 2.22: *Internal representation of the initial specific capacities*

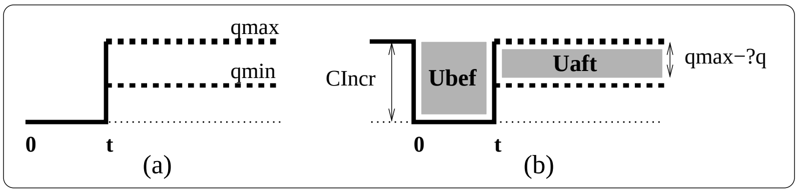


Figure 2.23: *Equivalent resource profiles for a* produce *proposition (a), and for its translation* $< u_{bef}, u_{aft}, C_{Incr} >$ *(b)*

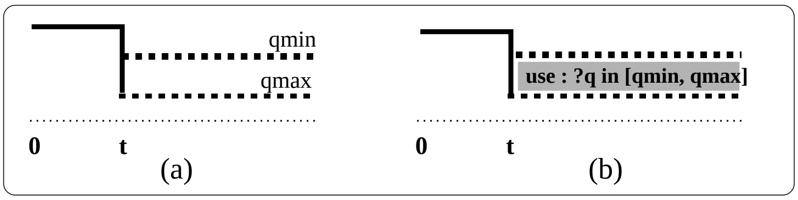


Figure 2.24: *Equivalent resource profiles for a* consume *proposition (a), and for its translation into a* use *proposition (b)*

Resource conflicts are detected as over-consuming sets of potentially overlapping borrowing propositions of the form:

$$\begin{aligned} &use(R(?r) :?q, (t_1, t_2)),\\ &\text{with } ?r \in D'_R \subseteq D_R\\ &\text{and } ?q \in [q_{min}, q_{max}]. \end{aligned}$$

As done before, the *consume* and *produce* propositions are translated into equivalent *use* propositions at the compilation of the domain model. However, the previous rewriting rule of a *produce(R():q,t)*, as a borrowing proposition *use(R():q,(0,t))* along with an increase of the capacity, does not hold anymore. Rewriting a production as a usage becomes more complicated due to the introduction of the two variables $?r$ and $?q$.

First, we need to limit the flexibility of the production representation: the resource produced by an action is allocated in order to be able to decide which specific capacity is increased by the action. Second, if $?q$ is not instantiated, it is not possible to decide by how much this specific capacity should be incremented. This problem is solved by the following rewriting rule:

$$\begin{array}{l} produce(R(r_P) :?q, t)\\ ?q \in [q_{min}, q_{max}] \end{array} \Leftrightarrow \begin{cases} u_{bef} = use(R(r_P) : q_{max}, (0, t))\\ u_{aft} = use(R(r_P) : q_{max} - ?q, (t, +\infty))\\ C_{Incr} = q_{max} \end{cases}$$

The rewriting rule for a *consume* proposition is straightforward: the consumption of a quantity $?q$ at time t is equivalent to a borrowing of the same quantity between t and the end of the horizon.

$$consume(R(?r) :?q, t) \Leftrightarrow use(R(?r) :?q, (t, +\infty))$$

Figures 2.23 and 2.24 present these rules in a graphical way.

The constant capacity increment C_{Incr} is associated with the proposition u_{bef}. During the search, the current capacities of the allocated resources in the partial plan P are computed as:

$$\forall r_i \in D_R, Capa(r_i) = C_R + \sum_{u_{bef}(r_i) \in P} C_{Incr}.$$

The plan contains a set of partially ordered and partially instantiated *use* propositions. It is not coherent w.r.t. resources if, in one of its interpretations, the cumulative usage of a set of overlapping use propositions on the same resource allocation exceeds its specific capacity. We now define a resource conflict in a partial plan as:

Definition 2.4.1 (resource conflict). *A* ***resource conflict*** *is a set of* use *propositions* $U = \{u_1, \ldots, u_k\}$ *present in the current partial plan* $\mathcal{P}$ *such that, if* $u_i = use(R(?r_i) : ?q_i, (st_i, et_i))$ *with* $?r_i \in D_{?r_i}$ *and* $?q_i \in [q_{i_{min}}, q_{i_{max}}]$*:*

1. the propositions potentially overlap ($\mathcal{P}$ *does not contain one of the constraints* $\{(et_i < st_j)_{i \neq j}\}$*); and*

2. the propositions potentially involve the same resource allocation: the variables $\{?r_i\}_{i=1..k}$ *can be unified in a unique equivalence class* $?r_U$*, with a non-empty valuation domain* $D_U = \bigcap_{i=1..k} D_{?r_i}$*; and*

3. there is at least one allocated resource $r \in D_U$ *such that* $\sum_{i=1}^{k} q_{i_{max}} > Capa(r)$*.*

2.4.2 Conflict detection

The analysis of a partial plan still consists in detecting Minimal Critical Sets which can be solved by the insertion of one resolver.

We slightly modified the detection algorithms described in section 2.3.3, which search for the smallest over-consuming cliques in a Possible Intersection Graph. One method could have been to consider each allocated resource separately. In that case, the PIG computed for an allocated resource r takes into account only propositions whose attribute's argument can be unified with r, and the clique search tree is developed for each graph as detailed before. This approach however can lead to useless clique computations: a clique U with $D_U = \{r_1, r_2\}$, for instance, would be considered twice, in the graphs corresponding to r_1 and to r_2.

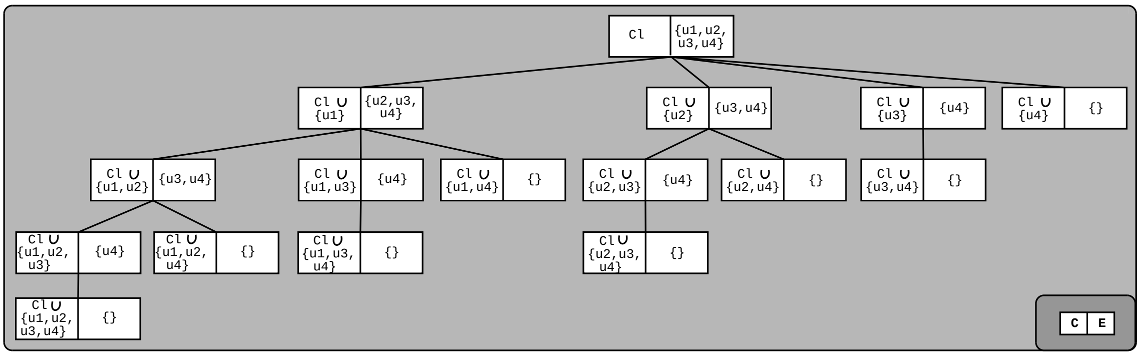


Figure 2.25: *Example of enlargement tree*

Instead, we associate a PIG with the resource type R. As explained before, the nodes of this PIG are the *use* propositions on the resource attribute R and an edge between two nodes exists if the corresponding propositions overlap in some temporal instantiation of the plan. A similar search tree is developed to find the cliques of the PIG. A node is defined by the same triplet $< \Delta(N), P(N), \Phi(N) >$. This time however, $\Phi(N)$ is computed by searching the minimal critical sets U such that:

- they have at least one proposition in each $\Delta(N_f)$, $N_f \in fathers(N) \cup N$,
- the attributes' arguments of all propositions can be unified in one equivalence class $?r_U$, with the valuation domain D_U,
- $\sum_U q_{max} > min_{r \in D_U}(Capa(r))$,
- the conflict is minimal.

The principle of the enumeration method used to compute $\Phi(N)$ consists in 3 steps:

1. compute the set of cliques CLIQUES having one proposition in each $\Delta(N_f)$, $N_f \in fathers(N) \cup N$, and such that their arguments can be unified in one equivalent variable $?r_C$;
2. progressively enlarge each clique $Cl \in$ CLIQUES by adding one proposition from $Pool$, where $Pool$ is made of the propositions belonging to the sets $\{\Delta(N_f)\}_{N_f \in fathers(N) \cup N}$ whose arguments can be unified with $?r_C$;
3. add a clique to $\Phi(N)$ if it is over-consuming and a minimal conflict.

Nevertheless, it should be noted that the capacity criterion used to detect an over-consumption now varies with the clique. Especially, if we consider a sub-clique $C' \subseteq C$, then $D_C \subseteq D_{C'}$, and $min_{r \in D_{C'}}(Capa(r)) \leq min_{r \in D_C}(Capa(r))$. This property has an impact on the performance of the second step of this enumeration method (i.e. the clique enlargement) and on the verification that a conflict is minimal. Thus, the procedure originally implemented to achieve the clique enlargement has been complemented by a second one.

The enlargement process can be viewed as the development of an ordered tree (cf. Figure 2.25), whose nodes are defined by the pair $< C, E >$, where C corresponds to a clique to enlarge, and E corresponds to the set of propositions that can be used to do so, sorted by their consumption value, starting at the most consuming. A node $M =< C, \{u_1, \dots, u_k\} >$ has k children $\{M^i\}_{i=1..k}$ computed as:

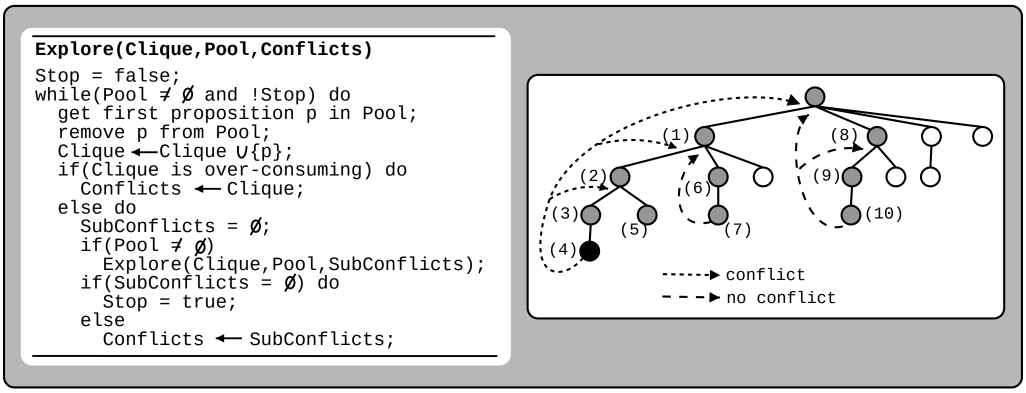
$$\begin{cases} M^i =< C', E' >, \\ C' = C \cup \{u_i\}, \\ E' = \{u_{i+1}, \dots, u_k\}. \end{cases}$$

The root node is equal to $M_0 =< Cl, Pool >$ and each time a clique is a MCS, it is added to $\Phi(N)$. The analysis process looks for all smallest MCS, and it is not necessary to fully develop this tree. The two procedures differ by their exploration strategy. The first one is more efficient if the conflicts are situated in the depths of the tree. It is employed when Cl and $Pool$ concern propositions whose resource attributes are all fully allocated. The second one is used otherwise.

First procedure

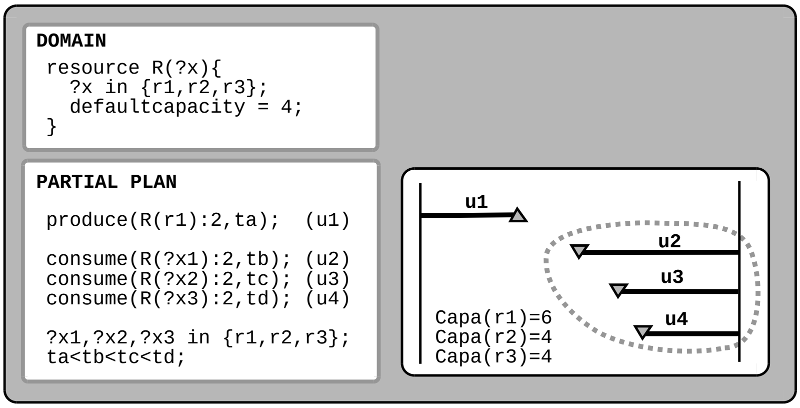
The tree is developed following a depth-first strategy with some pruning. A branch is explored so far as to find a conflict. In that case, the next branch is explored in the same way. If a conflict-less leaf is reached, the information is reported back to the root node of the branch and the following branches starting from this node are pruned. This pruning is justified by the inclusion of the following cliques in the cliques found in the branch. Thus, the next sub-cliques will not contain any other conflict. This property however does not hold if the resource attributes are not all fully allocated since the capacity criterion $min_{r \in D_C}(Capa(r))$ possibly decreases when the size of the clique decreases.

Figure 2.26 presents the recursive function that implements this strategy, as well as an example of exploration of the tree described in Figure 2.25. In this example, the tree only contains the conflict $Cl \cup \{u_1, u_2, u_3, u_4\}$ (the black node). The numbers indicate the order in which the nodes have been computed, the white nodes have been pruned.



```
Explore(Clique,Pool,Conflicts)
Stop = false;
while(Pool ≠ ∅ and !Stop) do
  get first proposition p in Pool;
  remove p from Pool;
  Clique ←— Clique ∪{p};
  if(Clique is over-consuming) do
    Conflicts ←— Clique;
  else do
    SubConflicts = ∅;
    if(Pool ≠ ∅)
      Explore(Clique,Pool,SubConflicts);
    if(SubConflicts = ∅) do
      Stop = true;
    else
      Conflicts ←— SubConflicts;
```

Figure 2.26: *Tree exploration with the first procedure*



```
DOMAIN
 resource R(?x){
   ?x in {r1,r2,r3};
   defaultcapacity = 4;
 }
```

```
PARTIAL PLAN
 produce(R(r1):2,ta);  (u1)

 consume(R(?x1):2,tb); (u2)
 consume(R(?x2):2,tc); (u3)
 consume(R(?x3):2,td); (u4)

 ?x1,?x2,?x3 in {r1,r2,r3};
 ta<tb<tc<td;
```

Figure 2.27: *Example of conflict solved by separation or domain reduction*

Second procedure

The second procedure explores the tree following a breadth-first strategy with some pruning. A node M^i is computed if the argument of u_i can be unified with $?r_C$, and the tree is developed as long as the size of the new clique remains less than or equal to the size of the smallest MCS found so far by the analysis process. This method guarantees that, when a conflict is found, it is minimal.

2.4.3 Resolvers

To guarantee the completeness of the search process, we need to propose additional resolvers besides the **ordering** constraints and the **action insertions** presented in section 2.3.3. In this subsection, we detail these new resolvers, their costs, and consider how we can reduce the branching factor by sorting the resource conflicts in different classes, for which some of the resolvers can be discarded.

New resolvers

Let us consider an MCS $U = \{u_1, \dots, u_k\}$, with $u_i = use(R(?r_i) : ?q_i, (st_i, et_i))$, $?r_i \in D_{?r_i}$, $?q_i \in [q_{i_{min}}, q_{i_{max}}]$. The variables $\{?r_i\}_{i=1..k}$ can be unified in the equivalence class $?r_U$ corresponding to the valuation domain $D_U = \bigcap_{i=1}^{k} D_{?r_i}$. And the set U is over-consuming w.r.t. the capacity criterion $Capa_U = min_{r \in D_U}(Capa(r))$.

- An obvious resolver, called **separation**, consists in posting one of the following differentiation constraints $\{(?r_i \neq ?r_j)\}_{(i,j)\in[1..k]\times[1..k], i\neq j}$.
- Another resolver, called **domain reduction**, can be applied if for some allocated resource $r \in D_U$, the set U is not a conflict. We partition D_U into $D'_U \cup D''_U$, where

D'_U corresponds to the allocations for which U is a conflict, and D''_U corresponds to the allocations for which U is not a conflict. The resolver consists in selecting one proposition u_j and removing the set D'_U from the valuation domain of $?r_j$. The cost of this resolver is computed using the formula:

$$cost(\text{domain reduction}) = \frac{card(D'_U)}{card(D_{?r_j})} * c_R,$$

where c_R is a coefficient tuned by the user. The conflict $\{u_2, u_3, u_4\}$ in Figure 2.27 can be solved by the following disjunction of resolvers: $(?x_1 \neq ?x_2) \vee (?x_1 \neq ?x_3) \vee (?x_2 \neq ?x_3) \vee (?x_1 \notin \{r_2, r_3\}) \vee (?x_2 \notin \{r_2, r_3\}) \vee (?x_3 \notin \{r_2, r_3\})$.

- The following resolver, called **quantity limitation**, consists in reducing the domains of the resource usages so as to nullify the over-consumption: $OverConso = \sum_{i=1}^{k} q_{i_{max}} - Capa_U$. This resolver is applicable if $\sum_{i=1}^{k} q_{i_{min}} \leq Capa_U$ holds, and amounts to posting the constraint:

  $$\sum_{i=1}^{k} ?q_i \leq Capa_U.$$

  The global quantity consumed by the propositions ranges in an interval whose length is: $Q = \sum_{i=1}^{k}(q_{i_{max}} - q_{i_{min}})$, and the cost of the resolver is computed as:

  $$cost(\text{quantity limitation}) = \frac{OverConso}{Q} * n * c_L.$$

  Once again, c_L is a parameter tuned by the user.

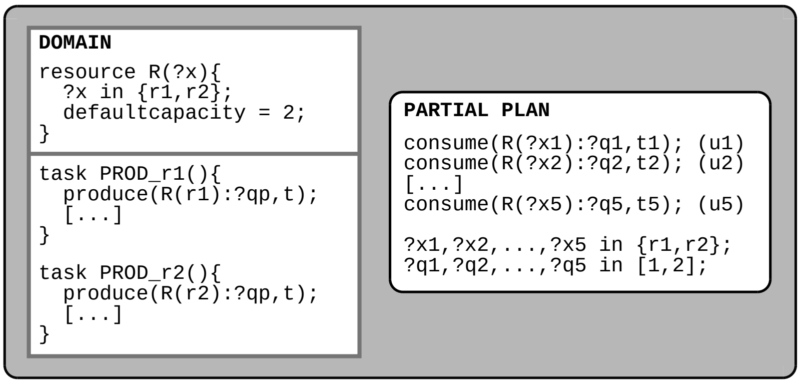
- Finally, the resolver presented in section 2.3.3, that consists in inserting an action that produces a quantity $?q_p$ of the resource is complemented with the constraint:

  $$?q_p \geq \sum_{i=1}^{k} ?q_i - Capa_U.$$

Classes of resource conflicts

The result of the resource analysis is the set of the smallest MCS with their associated disjunction of resolvers. This can lead to a quite important branching factor. Hopefully, it is possible to detect and discard early during this analysis phase some resolvers which will lead to dead-end branches, or some resolvers which lead to useless constraints.

Consider for instance the simple partial plan described in figure 2.28 and containing five *consume* propositions on the resource attribute $R(?x)$. The resource analysis process will detect and propose resolvers for all conflicts of the minimal size (i.e. 2 propositions). The



```
DOMAIN
resource R(?x){
  ?x in {r1,r2};
  defaultcapacity = 2;
}
task PROD_r1(){
  produce(R(r1):?qp,t);
  [...]
}
task PROD_r2(){
  produce(R(r2):?qp,t);
  [...]
}
```

```
PARTIAL PLAN
consume(R(?x1):?q1,t1); (u1)
consume(R(?x2):?q2,t2); (u2)
[...]
consume(R(?x5):?q5,t5); (u5)

?x1,?x2,...,?x5 in {r1,r2};
?q1,?q2,...,?q5 in [1,2];
```

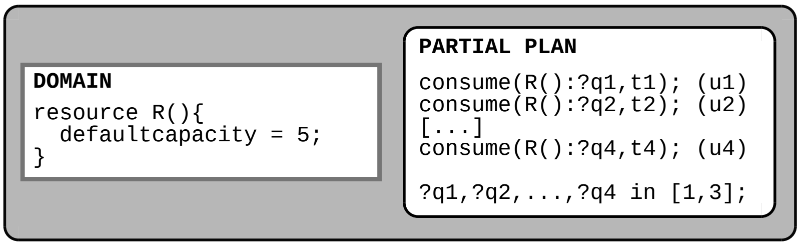
Figure 2.28: *Example of type-conflict*

MCS $\{u_1, u_2\}$ for instance can be solved by $(x_1 \neq x_2) \vee (?q_1 + ?q_2 \leq 2) \vee (PROD\_r1()) \vee (PROD\_r2())$. But the two first resolvers appear to be useless as the total resource capacity is 4 whereas the minimal consumption in the plan is 5. Therefore, *separation* and *quantity limitation* can be discarded from the disjunction of resolvers in certain cases.

We implemented such a strategy by doing a pre-analysis for each resource type R, with the possible allocations D_R, and the initial capacity C_R. Using the same algorithm as for the *MCS* detection, but without considering resource parameters, we search for the **type-conflicts** $U_T = \{u_1, \ldots, u_m\}$ corresponding to cliques such that no ordering constraint is possible and:

$$\sum_{i=1}^{m} q_{i_{min}} > C_R * card(D_R) + \sum_{u_{bef}(r) \in Plan, r \in D_R} C_{Incr}$$

Then, during the conflict detection process, we test if the *MCS* is included in at least one *type-conflict*. In that case, the set of resolvers is reduced to the possible *action insertions*.



```
DOMAIN
resource R(){
  defaultcapacity = 5;
}
```

```
PARTIAL PLAN
consume(R():?q1,t1); (u1)
consume(R():?q2,t2); (u2)
[...]
consume(R():?q4,t4); (u4)

?q1,?q2,...,?q4 in [1,3];
```

Figure 2.29: *Example of resource contention solved by a* maximal quantity limitation *resolver*

Another type of simplification can be deduced from the resource conflict analysis. Consider now the example presented in Figure 2.29. The analysis process will detect the smallest MCS $\{u_i, u_j\}_{(i,j) \in [1..4] \times [1..4], i \neq j}$ and propose for each one a disjunction of resolvers, reduced in this example to the deterministic *quantity limitation* $?q_i + ?q_j \leq 5$. The planning process will successively solve the conflicts of size 2, then handle the conflicts of size 3 (solved by a constraint $?q_i + ?q_j + ?q_k \leq 5$), etc.

We can note however, that this partial plan could have been fixed in one step, by the insertion of the constraint $?q_1 + ?q_2 + ?q_3 + ?q_4 \leq 5$. Thus we propose a new resolver, called **maximal quantity limitation**, and only applicable to MCSs that satisfy the following conditions: the propositions necessarily overlap and the arguments are unified.

If such an MCS is found, the *Necessary Intersection Graph*[22] is computed, as well as the set of its maximal cliques $\{U_{M_i}\}_{i=1..l}$ that contain the MCS. The disjunction of resolvers for the MCS includes l *maximal quantity limitations*, each one corresponding to a *quantity limitation* constraint posted for a maximal conflict U_{M_i}. The algorithms used to compute the maximal cliques are detailed in Appendix 1.

Finally, the MCS are sorted into different classes, summarized in Table 2.6. To each class corresponds a set of potentially applicable resolvers.

Characteristics of the MCS	**Classes**		
	C_1	C_2	C_3
Is contained by a type-conflict	Y	N	N
Propositions necessarily overlap and arguments are unified	/	Y	N
Potential Resolvers	action insertion	action insertion maximal quantity limitation	action insertion ordering separation domain reduction quantity limitation

Table 2.6: *Classes of MCS*

2.4.4 Conclusion

In conclusion, the extension of the expressiveness of the resource usages to variable quantities is very useful, both to represent more realistic models and to provide execution flexibility. Its main impact concerns the possible resolvers and the branching factor linked to a resource conflict.

The execution flexibility is illustrated in Figure 2.30 for the resource STORAGE con-

[22] The nodes of the NIG are the *use* propositions on the resource attribute, and an edge between two nodes exists if their corresponding propositions necessarily overlap and if the attribute's arguments of both propositions are unified.

sumed and produced by the plan in Figure 2.20. The higher part represents the *use* propositions and their variable quantities along with their valuation domains. In this particular example, their timepoints are totally ordered. The lower part represents the possible variations of the resource level over the plan horizon, and the least committed constraints contained by the plan that guarantee that this level never becomes negative nor exceeds the maximal capacity.

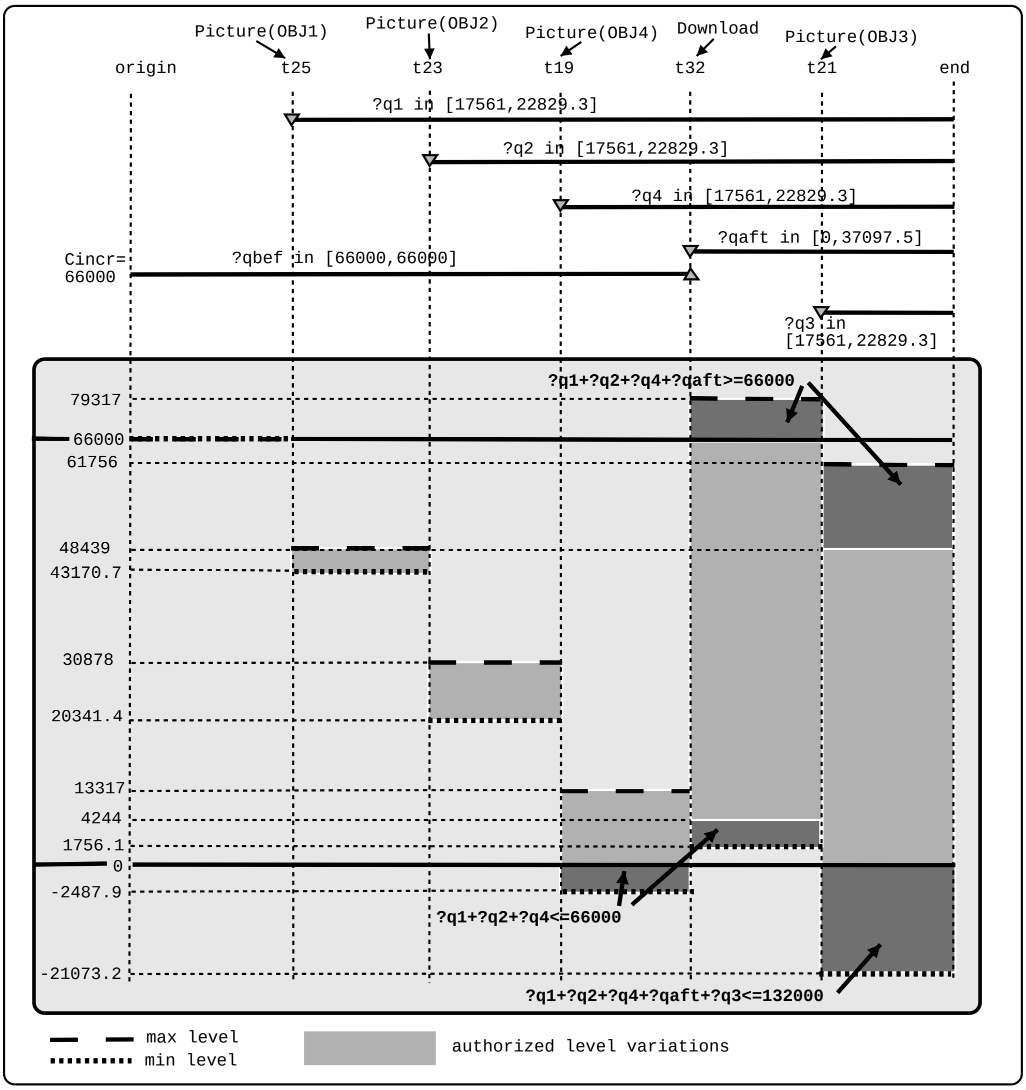


Figure 2.30: *Resource usage flexibility for the plan example in Figure 2.20*

On the other hand, the extension to diverse resource allocations is less useful and makes the resource detection process even more expensive (test of potential unification of the variables, etc.)

Chapitre 3

Ce chapitre décrit l'approche que nous avons mise en place pour exécuter un plan temporel et réagir aux événements imprévus. Cette approche exploite la flexibilité et le parallélisme des plans produits par IxTeT. Elle combine planification, réparation de plan et contrôle d'exécution en tenant compte tout particulièrement des contraintes temporelles et des mises à jour des niveaux de ressource.

IxTeT-eXeC est constitué :

- d'un exécutif temporel, TeXeC, qui interagit avec le système contrôlé,
- du planificateur IxTeT modifié pour prendre en compte le contexte d'exécution.

Les deux processus agissent sur la même base de données : un plan initial produit par IxTeT puis "déroulé" par TeXeC selon un cycle "Perception / Réparation de plan / Action". Lorsque l'intégration des messages dans le plan l'invalide partiellement, une réparation peut être menée en parallèle de l'exécution des parties valides. Dans certains cas cependant, le cycle doit être arrêté et une replanification complète est requise.

La première section décrit l'exécutif temporel : son rôle, ses interactions avec le système et les stratégies employées dans les différents cas d'échec. La deuxième section définit sous quelles conditions un plan peut être simultanément exécuté et adapté. La troisième section présente les principes sur lesquels le cycle d'exécution repose. Les sections 3.4 à 3.6 détaillent les trois phases du cycle. La septième section décrit la stratégie de replanification. Le chapitre se termine sur une discussion des avantages et améliorations possibles du système.

L'exécutif temporel

L'exécutif temporel remplit les fonctionnalités suivantes :

- il contôle le réseau temporel du plan pour décider de l'exécution des actions,
- il intègre dans le plan les bilans d'état retournés à la fin de chaque action,
- il surveille la durée des actions et réagit lorsque les contraintes temporelles ne sont pas satisfaites,
- il réagit aux échecs d'action, aux nouveaux buts et à des modifications soudaines de capacité des ressources,
- il lance et contrôle les deux processus d'adaptation de plan (réparation / replanification).

TeXeC envoie deux types de commande (lancement/interruption des actions si elles sont préemptibles). Il reçoit par ailleurs trois types de message :

- un bilan à la fin de l'exécution de chaque action contenant le statut de l'action et l'état du système (notamment le niveau des ressources),

- une notification de la modification de la capacité d'une ressource,
- un nouveau but.

L'exécution se base sur le réseau temporel. Une stratégie particulière d'exécution est associée à chaque type de timepoint (timepoints de début/fin d'une action, d'un but, d'une proposition contingente, etc.).

Pendant l'exécution d'un plan, les situations qui nécessitent l'adaptation du plan sont:

- l'arrivée d'un nouveau but,
- une mise à jour du niveau des ressources impliquant un conflit futur,
- l'échec d'une action,
- les échecs temporels (un bilan est reçu trop tôt ou trop tard ...).

Définition d'un plan exécutable

Dans cette section, nous précisons la définition d'un plan partiel partiellement exécuté jusqu'à l'instant t, et détaillons comment est effectué le calcul du niveau théorique d'une ressource à t. Nous définissons ensuite sous quelles conditions un plan et ses différents objets (timepoint, action, but) sont exécutables.

Cycle "Perception/Réparation de plan/Action"

TEXEC se réveille lorsque un message a été reçu, lorsque il est temps d'exécuter un timepoint ou lorsque une réparation de plan est en cours. La réparation de plan correspond à une recherche dans l'espace des plans partiels distribuée sur plusieurs cycles : une portion limitée de la durée du cycle est allouée à la réparation pour permettre l'exécution d'actions et la réception de messages. La planification est reprise dans la phase Réparation du cycle suivant.

Cet entrelacement de la planification et de l'exécution peut être poursuivi tant qu'un plan *exécutable* (réseaux de contraintes consistants et actions en cours supportées par le plan) est disponible à la fin de chaque phase du cycle. Autrement, le cycle est interrompu pour replanifier.

Perception

Nous détaillons pour les trois types de message comment le message est intégré dans le plan et comment il peut partiellement l'invalider (en introduisant des sous-buts à établir et/ou des conflits de ressource). Si une réparation du plan est nécessaire, il faut de plus rompre certains liens causaux pour permettre l'insertion de nouvelles actions.

Réparation de plan

La distribution de la planification sur plusieurs cycles pose deux problèmes :

- Sur quel plan repose l'exécution dans la phase "Action" du cycle, surtout si aucune solution n'a été trouvée?
- A partir de quel plan et de quel arbre de recherche se poursuit le processus de planification dans le cycle suivant?

Action

TEXEC exécute les timepoints dont l'échéance arrive avant la fin du cycle. L'exécution d'un timepoint varie selon le type de timepoint et est éventuellemnt retardée si l'objet correspondant (action/but) n'est pas exécutable lorsque un processus de réparation est en cours.

Replanification complète

Un plan initial est composé à partir de l'état courant du système et des buts pas encore réalisés. La difficulté principale vient de l'incertitude sur la durée de la planification : il faut pouvoir garantir qu'à la fin de la planification il reste suffisamment de temps pour exécuter le plan solution tout en respectant les rendez-vous temporels.

Discussion

La réparation de plan est plus efficace pour des applications dont les plans sont temporellement flexibles et avec un fort parallélisme (des actions pouvant s'exécuter indépendamment les unes des autres).

Les performances du système pourraient être améliorées en tenant compte de la contingence des durées, avec des plans dont le réseau temporel vérifie les propriétés de pseudo contrôlabilité et contrôlabilité dynamique.

Chapter 3

IxTeT-eXeC: Interleaving temporal planning and plan execution

The previous chapter detailed the principles and mechanisms of an STN-based POCL planner and scheduler. We now consider how we can exploit the parallelism and flexibility of the plans produced by IxTeT to deal with many types of execution events.

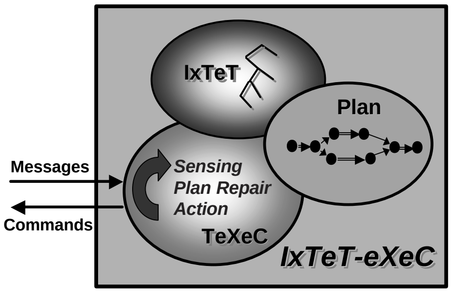


Figure 3.1: *Components*

We propose a new framework to combine deliberative planning, plan repair and execution control that takes into account resource level updates and temporal constraints. These processes are embedded thanks to two components (cf. Figure 3.1):

- a temporal executive, called TeXeC, interacting with the controlled system (e.g. through a procedural executive);
- the temporal planner IxTeT, modified to take into account the execution context.

Both components reason about and act upon the same plan database. A first solution plan is produced by IxTeT. This plan is then run by the temporal executive following a "Sense/Plan Repair/Act" cycle. The executive wakes up when it needs to do something, i.e. a message has been received, or it is time to execute some timepoint or a plan repair process is in progress.

Let us call *ExecutingPlan* the plan currently under execution. *Sensing* consists in integrating messages in *ExecutingPlan*, some of which may partially invalidate it. If *ExecutingPlan* contains new flaws, a *plan repair* consists in keeping the structure of the plan and taking advantage of the temporal flexibility to try to find a solution plan. Planning is distributed on several cycles and thus interleaved with execution to allow reactivity to events and concurrent execution of the valid part. *Acting* consists in determining which timepoints have to be executed, processing them or detecting time-out.

Under certain circumstances, this cycle must be stopped and a complete replanning is mandatory. The execution of the current plan is aborted, and a new plan is generated from the description of the current state and the set of not yet achieved goals.

In section 3.1, we briefly introduce the temporal executive: its role, its interactions with the controlled system and the strategies proposed to handle events (new goals and execution failures). In section 3.2, we formally define under which conditions a plan can be simultaneously executed and adapted. In section 3.3 we present the principles on which the execution cycle relies. In sections 3.4 to 3.6, we explicit the three phases of the cycle: sensing, plan repair, action. In section 3.7, we describe the replanning strategy. We finish with a discussion on the advantages and possible improvements of the system.

3.1 Temporal executive

3.1.1 Overview

The key component in IxTeT-eXeC is the **temporal executive** TeXeC, which interacts with the planner and the controlled system. In our experiments, TeXeC is interfaced with a procedural executive (OpenPRS) that allows interactions with the user and with the functional modules of the robot (see Chapter 4 for a thorough description of this). The plan execution is controlled by both executives as follows. TeXeC decides when to start or stop an action in the plan and handles plan adaptations. OpenPRS expands the action into commands to the functional modules, monitors its execution and can recover from specific failures. It reports to TeXeC upon the action completion and also transmits the user requests.

We further detail the functionalities of the temporal executive:

- It controls the temporal network of the plan to decide the execution of actions (launch/stop) and maps the timepoints to their real execution time.
- It integrates in the plan the state reports (including resource levels) sent by the controlled system upon each action completion.
- It monitors the duration of actions and reacts when temporal constraints are not satisfied.
- It reacts to action failures reported by the controlled system, as well as to requests to insert a new goal and to sudden alterations of resources capacity.
- It starts and controls the processes of plan adaptation (plan repair and replanning).

One remark can be drawn from this list. At planning level, the model represents the world as a set of abstract state variables, functions of time, generally piece-wise constant. Thus, the executive actively monitors the temporal constraints of the plan, the effects of actions but not their progress. The monitoring of the internal state of the controlled system is left to the procedural executive and the more informed lower levels of the architecture[1].

3.1.2 Interactions with the controlled system

IxTeT-eXeC sends commands to and receives messages from the procedural executive.

As presented before (cf. section 2.1.6), an action in the plan is characterized by: its name a, its grounded parameters p_a, its starting timepoint st^a, its ending timepoint et^a, an identifier i_a and a label denoting its execution behavior ($earlyPreemptive/ latePreemptive/nonPreemptive$).

IxTeT-eXeC starts the execution of an action by sending the command (LAUNCH a p_a i_a) to the controlled system. If the action is controllable and did not already terminate by itself, the command (END i_a) is sent to the system as soon as possible if the action is $earlyPreemptive$, as late as possible if it is $latePreemptive$. If the action is non preemptive, IxTeT-eXeC just monitors if it is completed in due course.[2]

IxTeT-eXeC receives three types of messages: a report upon an action completion, a notification of the alteration of a resource capacity, or a new goal[3].

- An action report has the following form:

  (REPORT i_a status <logical state> <resource levels>)

  It necessarily contains the ending status of the action, i.e. either *nominal*, *interrupted* or *failed*. If the action model specifies some resource usage, the report contains the level of the allocated resource at the end of the action. Note that the system returns a global level of a resource and not a report on the specific quantity used by the action. Indeed, in case of concurrent actions, it is often impossible to discriminate the exact share of a resource usage due to each action. If the ending status is not nominal, the report also contains a partial description of the logical state of the system, i.e. the final values of the grounded logical attributes which are relevant to the action.

[1] The Requests control level of the LAAS architecture can perform such monitoring.

[2] The difference between a late preemptive action and a non preemptive one relies in the capacity for the system to actually command the stop of the action or not. This label does not take into account the possible contingency of an action duration, and a late preemptive action may be stopped even though its effects are not completely achieved (thus requiring plan update and adaptation). We will consider in section 3.8 how the non-controllability of the duration links could be taken into account to improve the overall performances.

[3] This goal may come from the user or from the controlled system itself, for instance when an on-board image processing reveals new interesting targets.

- A message notifying a capacity alteration has the following form:

  (CAPACITY Res q +/-)

  It indicates that the capacity of the allocated resource Res has increased or decreased by a certain quantity q. For instance, this type of message will inform TeXeC of a partial loss of memory storage, or of the sudden unavailability of a device.

- A new goal is specified by:

  (GOAL Att Val Prio Dur Orig MinAchiev)

  A goal corresponds to a new *hold* proposition on the grounded attribute Att set to the fixed value Val. The other data define the priority of the goal (Prio), the duration of the *hold* proposition (a real interval Dur), a temporal constraint between the origin timepoint of the plan and the starting timepoint of the goal (Orig), and an estimation (MinAchiev) of the minimal duration required to achieve the goal (possibly null, if no estimation is available).

3.1.3 Temporal network execution

TeXeC bases its control on the temporal network of the plan. This network is an STN. The constraints are propagated during the planning process by quite efficient algorithms that keep the network minimal (cf. section 2.2.1). During the execution process, these same algorithms are used to propagate the actual occurrence time of timepoints. If the plans are big and the number of timepoints becomes significant, it could be interesting to use some more efficient execution algorithms like the one proposed in [Tsamardinos 98] and based on a filtering of non-dominating edges and local propagation in the filtered network.

The executive discriminates between different types of timepoints:

- st^H and et^H, respectively the origin and the ending timepoint of the planning horizon;
- the starting timepoints of the actions (st^a);
- the ending timepoints of the actions (et^a), with a distinction between the ending timepoints of an early, late or non preemptive action;
- the starting and ending timepoints of the goal propositions (st^g and et^g);
- the timepoints of contingent propositions (which specify the evolution of contingent attributes in the initial scenario, cf. 2.1);
- miscellaneous timepoints, such as the intermediate timepoints used in the description of actions.

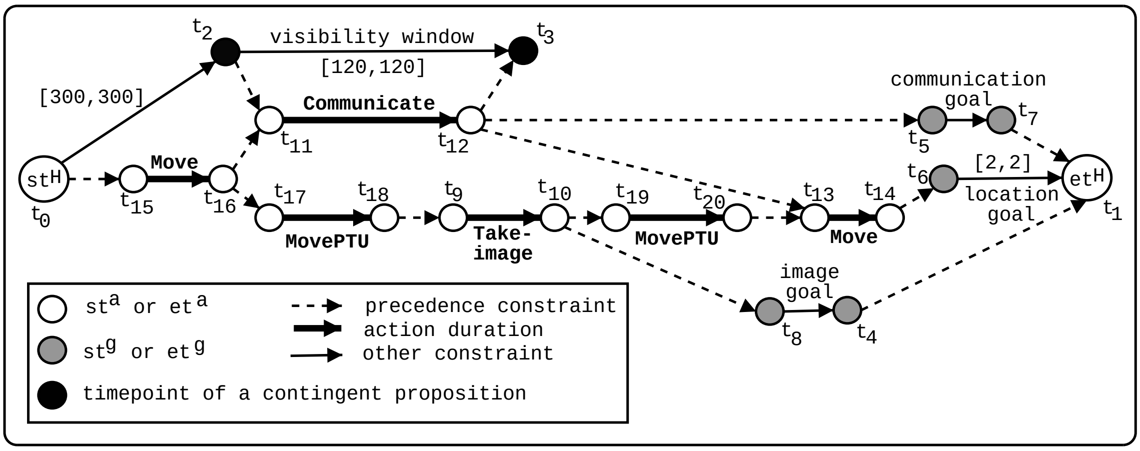


Figure 3.2: *Example of temporal plan network*

TEXEC decides when to execute a timepoint from the numeric constraints of the STN, and how to execute it from its type. Currently, the executive has a well defined execution strategy for the action and goal timepoints. Future work may improve the skills of the executive by allowing it, for instance, to check that contingent events actually occurred or to perform specific monitoring at the intermediate timepoints. For the moment, TEXEC executes the timepoints of contingent propositions by a simple instantiation to their execution time and skips the "miscellaneous" timepoints.

Figure 3.2 shows an example of temporal plan network for the rover domain[4]. The plan achieves 3 goals: one `image goal`, one `communication goal` with the ground station during the visibility window and one `location goal` (the rover is back at the lander at the end of the mission). We can notice an important characteristic of the type of STN produced by IXTET: the timepoints of an action are exclusively linked to the timepoints outer of the action by precedence constraints, more or less flexible according to the global constraints of the plan duration and horizon. This feature is induced by the use of a causal link structure during the planning process and provides some temporal flexibility, used during parallel planning and execution to postpone the launch of an action or to insert a new action.

Afterwards, we will call **execution interval** the time interval during which a future timepoint t should be executed to be consistent with the temporal constraints. This interval, noted $[t_{lb}, t_{ub}]$, corresponds to the domain of $(t - st^H)$.

An **execution time** t_{exec} is also associated with each future timepoint. It corresponds to the time at which the timepoint t should be executed and its value varies with the type of the timepoint (see section 3.6).

[4] The STN contains a link between each pair of timepoints. For clarity, this figure only represents the most relevant ones.

Finally, the **occurrence time** t_{occ} of a timepoint corresponds to its actual execution time, and the executive updates the STN by posting and propagating the constraint $(t - st^H) = t_{occ}$.

3.1.4 Non nominal situations and strategies of plan adaptation

A plan is produced and then executed by controlling its temporal network. We consider which situations may arise that forbid further execution of the plan without adapting it:

1. A **new goal** is requested.
2. **Resource level adjustment -** A resource conflict appears in the plan after the update of a resource level, or a sudden alteration of capacity. If an action has consumed/produced more/less than expected, the plan may contain future resource contention. If a reservoir-like resource is over-produced, it may contain future overflow.
3. **Action failure -** The controlled system returns a non nominal report.
4. **Temporal failures -** The temporal network constrains each timepoint to occur inside its execution interval. Thus two types of failure lead to an inconsistent plan: the event corresponding to the timepoint (typically the end of an action) happens *too early* or *too late* (time-out).

Actually we need to make a distinction between two types of time-out situations. In one case, some temporal flexibility remains in the rest of the plan. This situation may arise for instance when a non preemptive action takes more time than forecast in the action model. In the other case, the plan is temporally over-constrained (the links corresponding to precedence constraints are squeezed to the minimal possible duration). This situation may arise for instance when the starting timepoint of an action has been postponed for too long a time.

To take advantage of the temporal flexibility of the plan, the dynamic plan adaptation strategy has two steps. A first attempt consists in a **plan repair**. The plan has been partially invalidated by the integration of the failure context[5], its structure (the actions and their ordering) is kept as is, and the planner tries to restore the lost properties. The execution of the valid part is pursued in parallel. In certain conditions however (notably if the plan repair fails or if a timepoint times out), the execution must be aborted and a **replanning** process is started. Generally, this process searches for a new plan, applicable in the current state of the system and achieving the set of remaining goals. If such a plan is not found in due time, a goal is abandoned, and the replanning process is started again. However, in case of time-out with no temporal flexibility left, one goal is directly removed from the replanning problem.

[5] The plan may now contain open conditions and/or resource conflicts.

3.2 An executable plan

The plan repair and plan execution processes are interleaved and operate on the same plan database. This intertwining of partial order planning and execution may introduce new flaws in the plan, and we need to formally specify under which conditions such a partial plan remains executable. With this aim in view, we introduce some useful notations and definitions.

3.2.1 A partial plan partially executed up to time t

We extend the definition of a partial plan in the POCL framework presented in section 2.3.1 (page 57) to the definition of P_t: a **partial plan partially executed up to time t**.

Definition 3.2.1. $P_t = (RA_t, FA_t, S_t, G_t, Co_t, C_t, L_t, F_t)$

- RA_t is the set of currently **running actions** ($a \in RA_t$ if $st^a_{ub} < t$ and $et^a_{ub} > t$).
- FA_t is the set of **future actions** ($a \in FA_t$ if $st^a_{ub} \geq t$).
- S_t represents the **state of the system** at time t. It is composed of 2 sets corresponding to the logical state and to the resource state of the system:
  - $LgcS_t$ contains the last value of each grounded logical controllable attribute la present in the plan or, similarly in IXTET, the last executed *event* proposition for each la.
  - $RscL_t$ contains the level at time t of each allocated resource present in the plan.
- G_t is the set of **goals** not yet completely achieved at time t (and potentially not established). G_t contains goals such that $et^g_{ub} \geq t$.
- Co_t is more specific to IXTET. It contains the future evolution profile of grounded **contingent** attributes, i.e. the contingent propositions $hold(* : *, (st^h, et^h))$ and $event(* : (*, *), t^e)$ such that $et^h_{ub} \geq t$ and $t^e_{ub} \geq t$.
- C_t is the set of **constraints** on the variables appearing in FA_t, RA_t, S_t, G_t and Co_t.
- L_t is the set of **causal links** supporting future actions.
- F_t is the set of flaws (open conditions, threats, resource conflicts) present in the partial plan at time t.

Past actions do not belong to the plan anymore. Their variables (timepoints and parameters) have been instantiated and the corresponding constraints have been propagated in the constraint networks. Useful information about their effects is kept in S_t.

3.2.2 Level of an allocated resource r at time t in the partial plan P_t

The "theoretical"[6] level of a resource at a certain time in the future cannot be computed, since it depends on the partial order of the actions using this resource. But at time t the past part of the plan is completely instantiated and linearized. Figure 3.3 summarizes the effects of a past action on the level of a resource.

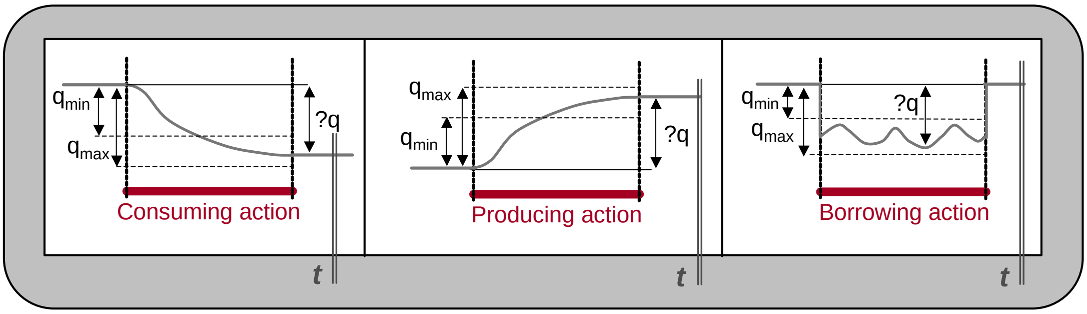


Figure 3.3: *Resource level after a past action*

The computation of the level at time t takes into account the global production or consumption of the resource over the past actions, as well as a worst case estimation of the global resource usage of the actions currently in execution. In fact, we need to consider two cases:

- *case (a):* if no running action modifies the allocated resource r, the exact level can be computed;
- *case (b):* if at least one action in RA_t requires the resource, only an estimation is available.

In case (a), the exact level is computed in IxTeT according to formula (3.1). We note $Capa(r)$ the capacity of the allocated resource. Let p be a production, belonging to the action a^p, of a quantity $?q_p$ of the resource r at time t^p and $P(r)$ be the set of productions of resource r in the plan. Similarly, $C(r)$ is the set of consumptions c, belonging to the action a^c, of a quantity $?q_c$ at time t^c, and $U(r)$ is the set of borrowings u, belonging to the action a^u, of a quantity $?q_u$ between st^u and et^u. Then, if no running action modifies r:

$$RscL_t(r) \quad = \quad Lev_t^{past}(r) \tag{3.1}$$

[6] i.e. according to the model of the actions and the constraints of the plan.

$$\text{with } Lev_t^{past}(r) \;=\; Capa(r) + \sum_{\substack{p\in P(r)/\\ t_{ub}^p<t \text{ and}\\ a^p\notin RA_t}} ?q_p - \sum_{\substack{c\in C(r)/\\ t_{ub}^c<t \text{ and}\\ a^c\notin RA_t}} ?q_c$$

This level is a variable ranging over $[lev_{lb}^{past}, lev_{ub}^{past}]$.

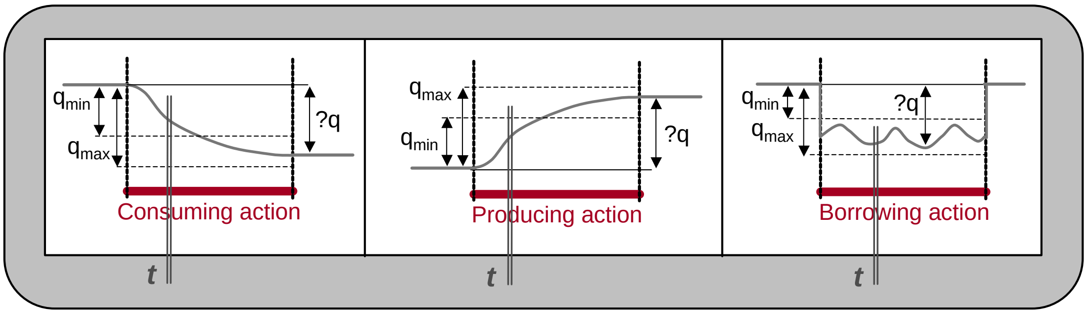


Figure 3.4: *Resource level during a running action*

In case (b), the uncertainty follows from the insufficient model of the resource usage (piece-wise constant whereas a production, for instance, may correspond to a monotonic increase). The previous level definition is completed by an estimation of the level modification induced by the running actions ($Lev^{RA}(r)$) according to formula (3.2). The total amount produced, consumed or borrowed by an action is represented by a variable $?q$ in $[q_{min}, q_{max}]$. At a given time in the course of the action, the only information that the planner can deduce is that the amount produced/consumed up to now is in $[0, q_{max}]$ or that the amount borrowed is in $[q_{min}, q_{max}]$ (cf. Figure 3.4). An estimation of the level would then be:

$$lev_{min}^{RA} \leq RscL_t(r) - Lev_t^{past}(r) \leq lev_{max}^{RA} \tag{3.2}$$

$$\begin{aligned} \text{with } lev_{min}^{RA} &= -\sum_{\substack{c\in C(r)/\\ a^c\in RA_t}} q_{c_{max}} - \sum_{\substack{u\in U(r)/\\ a^u\in RA_t}} q_{u_{max}} \\ lev_{max}^{RA} &= \sum_{\substack{p\in P(r)/\\ a^p\in RA_t}} q_{p_{max}} - \sum_{\substack{u\in U(r)/\\ a^u\in RA_t}} q_{u_{min}} \end{aligned}$$

Finally, the level of a resource r at time t is comprised in the interval $[RscL_{lb}, RscL_{ub}]$ with, in case (a):

$$\begin{aligned} RscL_{lb} &= lev_{lb}^{past} \\ RscL_{ub} &= lev_{ub}^{past}, \end{aligned}$$

and in case (b):

$$\begin{aligned} RscL_{lb} &= lev_{lb}^{past} + lev_{min}^{RA} \\ RscL_{ub} &= lev_{ub}^{past} + lev_{max}^{RA}. \end{aligned}$$

During the plan execution, this theoretical level is regularly compared with the value returned by the controlled system and adjusted (see section 3.4).

3.2.3 Executability

We now consider in which conditions the different objects of the plan are executable.

A timepoint in the temporal network is executed if it corresponds to a goal timepoint, to a starting or ending timepoint of an action or to a timepoint of a contingent proposition.

Definition 3.2.2 (executable timepoint). *A timepoint T is* executable *at time t if all timepoints T^p that must directly precede it in the temporal network have already been executed ($T^p_{lb} = T^p_{ub} < t$) and if t is contained by its execution interval.*

A goal corresponds to a property to achieve and eventually to maintain between st^g and et^g. A goal is satisfied if the associated *hold* proposition is established in the plan and not threatened by any other proposition. Thus:

Definition 3.2.3 (achievable goal). *A goal g is* achievable *at time t if st^g is executable and if $g \notin F_t$.*

Let A_t^f be the set of actions that are involved in the flaws of the plan at time t. The determination of A_t^f is straightforward in the case of open conditions and resource conflicts. In a threat case, an action a_k has effects in contradiction with the establishment of a proposition p by the causal link $a_i \xrightarrow{p} a_j$ and $(a_i \prec a_k \prec a_j)$ is consistent. A_t^f will contain a_k and a_j.

Definition 3.2.4 (executable action). *A future action a is* executable *at time t if its starting timepoint is executable and if $a \notin A_t^f$.*

Finally, we define an executable plan at time t as a plan that supports the actions currently in execution.

Definition 3.2.5 (executable plan). *A partial plan P_t is* executable *at time t if the constraint networks are consistent and if $RA_t \cap A_t^f = \emptyset$.*

3.3 "Sensing/Plan Repair/Action" cycle

The pseudo-code algorithm in Table 3.1 (page 111) complemented by the algorithms in Table 3.2 to Table 3.8 present, in a simplified version, how the execution cycle has been implemented in IxTeT-eXeC.

The executive determines, among the set of executable timepoints ($ExecutableTPs$), what should be the next one to execute and its execution time (t_{exec}). In fact, several timepoints may have to be executed during one cycle. The set of these timepoints ($ExecTPs$) is initialized with the set of timepoints whose execution time is t_{exec} and updated during the cycle (each time a timepoint is assigned its occurrence time) with the newly executable timepoints which have to be executed before the end of the cycle. Consider for instance the plan shown in Figure 3.2. If the executive wakes up to integrate a nominal report for the Take-image action, $st^{MovePTU}$ and $st^{image-goal}$ become executable and are executed in the current cycle.

A cycle is not necessarily composed of the three phases (Sense, Plan repair, Act). The executive may wake up just to send a command or to integrate a report. If a report partially invalidates the plan and requires a plan adaptation, two strategies can be applied. One, called **plan repair**, consists in keeping the structure of the plan (the future actions and their ordering) and try to restore its lost properties. The other one, called **replanning**, consists in interrupting the execution cycle and generating a new plan from scratch.

A plan repair process is distributed on several cycles: a limited share of the cycle duration is assigned to planning; when this time is reached, planning is stopped to allow possible action execution and/or message reception, and resumed in the immediately following cycle.

To interleave in such a way plan execution and plan adaptation, we need to guarantee a certain coherence of the plan P_t with respect to the execution context. We distinguish between two criteria of coherence. A local one is defined by the coherence between the planned actions and the executed ones at time t, which holds if:

1. Once an action is committed to execution, the planning process will not attempt to modify it.
2. The executive will not commit an action that may need to be modified by the planning process.

This criterion is complemented with a global one on the coherence of the temporal constraints over the plan horizon and the coherence of the atemporal constraints (such as resource quantities).

Thus planning and execution can be interleaved in cycles if the executive only launches actions which are executable and if the plan remains temporally and atemporally consistent and does not contain any flaw on the running actions. Finally, the execution cycle proposed in our approach relies on the following principle:

Cycling can be pursued as long as an *executable plan* is available at the end of each phase of a cycle. If this condition does not hold, cycling is interrupted and a complete replanning is mandatory.

The duration of the cycles determines the reactivity of the executive. Unlike the IDEA system [Muscettola 02] which guarantees a reaction time of two cycles by enforcing the planning process to complete and lead to a solution[7] within one cycle, in IxTeT-eXeC, the plan repair process is given as much time as allowed by the temporal flexibility left in the plan to find a complete solution plan. However, interleaving plan repair and plan execution allows to guarantee that an event (failure, new goal, etc.) will be integrated in the plan and taken into account in the planning problem and execution context within two cycles.

In IxTeT-eXeC, the duration of a cycle can not be known precisely. Its value mainly depends on the size of the plan and several factors have an influence, among them: the number of timepoints executed during the same cycle, the duration of the propagation in the STN, the duration of a message integration and plan invalidation in case of failure, the duration of the most expensive planning step, etc.

We call **timestep** the maximum time allowed to the execution cycle. It is defined by the user and may vary with the application. The uncertainty on the duration of the execution cycle has some consequences on the exact occurrence time of timepoints. If two timepoints have to be executed within an interval less than one timestep, the only guarantee is that they will be executed during the same cycle, according to their precedence constraints. This restriction imposes to design action models with a minimal duration greater than one timestep. We consider that a timestep of a few seconds is acceptable at this mission planning level.

Note also that the timestep is a worst case estimation: the cycle usually takes less time (especially if no plan repair is in process), and the global reactivity to new messages is better.

In the following sections, we further detail the three phases of a cycle and how the modifications made to the plan can leave it executable.

[7] A short planning duration is obtained by limiting the horizon of the planning problem (e.g. "what should be the next step?"). Execution failures can be repaired if a purely reactive model encompasses the reactions to all possible failure scenarios.

```
EXECUTION CYCLE

% $ExecutingPlan$: plan currently under execution
% $RunningActions$: set of actions currently under execution
% $RunningActionsWithThreat$: set of actions currently under execution
%                            which are threatened by the report of another action
% $st^{cycle}$, $et^{cycle}$: start and end time of the current cycle
% $ExecutableTPs$: the set of timepoints that are executable
% $t_{exec}$: execution time of the next executable timepoint
% $ExecTPs$: set of timepoints to execute during the cycle
% $WaitingExecTPs$: set of timepoints whose execution has been postponed
% $WaitingEndTPs$: set of ending timepoints for which an END command has been sent
%                  and the report has not yet been received
% $AbandonedGoals$: set of goals which have been abandoned during replanning attempts

$repair \leftarrow$ false
$abort \leftarrow$ false
$replan \leftarrow$ false
$newRepairSearchTree \leftarrow$ true
$noFlexTimeout \leftarrow$ false
initialize $ExecutableTPs$, $t_{exec}$, $ExecTPs$

cycle forever
    wake up if (current_time $\leq t_{exec}$) or ($repair$) or ($MsgQueue$ not empty)
        $st^{cycle} \leftarrow$ current_time
        $et^{cycle} \leftarrow st^{cycle} + timestep$
        if ($MsgQueue$ not empty)
            for each $Msg$
                if (Goal)
                    INTEGRATING_GOAL()
                if (CapacityChange)
                    INTEGRATING_CAPA()
                if (Report))
                    INTEGRATING_REPORT()
        if ($abort$) and ($RunningActions$ is empty)
           $replan \leftarrow$ true
        if ($repair$)
            PLAN_REPAIR()
        if ($replan$)
            REPLANNING()
        ACT()
        get next $t_{exec}$
        add executable timepoints which execution time is $t_{exec}$ to $ExecTPs$
end cycle
```

Table 3.1: *Execution Cycle*

3.4 Sensing

A message can be of three different types: new goal request, notification of a capacity alteration or report upon action completion. We consider in each case how the message integration can partially invalidate the plan. In that case and if a plan repair is required to fix open conditions and/or resource conflicts, the causal structure of the plan is further broken to allow the possible insertion of new actions.

3.4.1 Message integration

Goal request

```
INTEGRATING_GOAL()

% Goal($g$,$I^d$,$I^g$,$A^g$)
% $g$: hold(GoalAtt:GoalVal,($st^g$,$et^g$))
% $A^g = \{a \in \mathit{RunningActions}/g$ may threaten a proposition in $a\}$
% $\mathit{TemporalConstraints} = \{(et^g \prec et^H), (st^g > et^{cycle}), ((et^g - st^g)$ in $I^d), ((st^g - st^H)$ in $I^g)\}$
%                         $\cup\{(et^a \prec st^g)\}_{a \in A^g}$

if ($\mathit{TemporalConstraints}$ are consistent with $\mathit{ExecutingPlan}$)
    add $g$ and $\mathit{TemporalConstraints}$ to $\mathit{ExecutingPlan}$
    $\mathit{newRepairSearchTree} \leftarrow$ true
    if (!$\mathit{abort}$)
        REMOVE_CAUSAL_LINKS(GoalAtt,$st^g$)
        $\mathit{repair} \leftarrow$ true
else
    reject the goal
```

Table 3.2: *Integration of a "goal" message*

A goal is associated with two time intervals, one specifying the duration of the goal proposition (I^d) and one constraining the execution interval of the starting timepoint of the goal proposition (I^g). The insertion of a goal involves the creation of two timepoints st^g and et^g, constrained by:

$$\begin{aligned}
et^g \prec et^H &\text{ (before the end of the plan horizon)}\\
st^g > et^{cycle} &\text{ (after the end of the current cycle)}\\
(et^g - st^g) \text{ in } I^d &\text{ (duration)}\\
(st^g - st^H) \text{ in } I^g &\text{ (execution interval)}
\end{aligned}$$

To guarantee the plan executability, the goal proposition is also constrained to occur

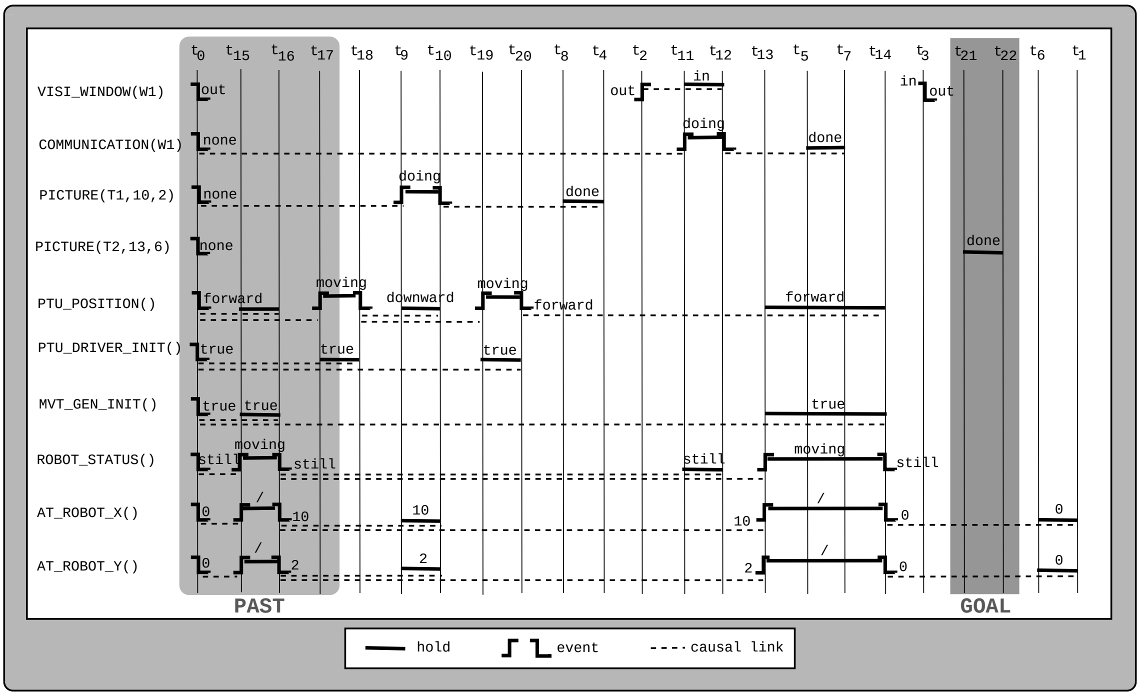


Figure 3.5: *Evolution of logical attributes and new goal*

after the end of each running action which contains a proposition on the goal attribute. If A^g represents the set of running actions which have a proposition possibly threatened by the goal proposition, then the goal ending timepoint must satisfy the following constraints:

$$\{(et^a \prec st^g)\}_{a\in A^g}$$

If one of the above constraints leads to an inconsistent plan, the goal is rejected. Otherwise, the plan contains an open condition to establish. Figure 3.5 represents the evolution[8] of the logical attributes in the plan example introduced in Figure 3.2. A flexible goal (GOAL Picture(T2,13,6) done 1 [2,2] [0,900] 0) has been received during the execution of the first MovePTU action, and inserted (timepoints t_{21} and t_{22}).

The establishment of a goal requires to break causal constraints on a subset of grounded logical attributes between et^{cycle} and st^g. In Figure 3.2 for instance, the goal establishment implies the insertion of Move, MovePTU and Take-image actions which have effects on the attributes AT_ROBOT_X(), AT_ROBOT_Y(), ROBOT_STATUS(), MVT_GENERATION_INIT(), PTU_DRIVER_INIT(), PTU_POSITION() and PICTURE(T2,13,6).

[8] The partially ordered future timepoints have been linearized in the figure for clarity purpose.

Capacity alteration

```
INTEGRATING_CAPA()

% CapacityChange(RsceAtt,$q$,+/−)

update the capacity
$newRepairSearchTree \leftarrow$ true
if (−) and (!$abort$) and (resource conflict exists)
    check executability of $ExecutingPlan$ % search for resource conflicts involving running actions
    if ($ExecutingPlan$ not executable)
        $abort \leftarrow$ true
        $repair \leftarrow$ false
    else
        REMOVE_CAUSAL_LINKS(RsceAtt,$et^H$)
        $repair \leftarrow$ true
if (+) and ($AbandonedGoals$ not empty)
    $abort \leftarrow$ true
    $repair \leftarrow$ false
```

Table 3.3: *Integration of a "capacity alteration" message*

As a consequence of an exogenous uncontrollable event, the capacity of a resource RsceAtt has decreased (-) or increased (+) by a quantity q. In IxTeT, such a modification is reflected on the plan by the insertion of a *consume* proposition (case (-)) or a *produce* proposition (case (+)) of the quantity q. This capacity update may induce future resource conflicts or on the contrary have serendipitous effects. In the current implementation of IxTeT-eXeC two strategies are followed.

1. If the capacity has decreased and the plan contains a conflict on *RsceAtt*, but is still executable, a plan repair is requested. As for a goal establishment, the repair of a resource conflict may require the insertion of actions to produce the resource. Likewise, causal links on a subset of grounded logical attributes have to be broken.

   To guarantee the executability of the plan, we need to find out that there is no resource conflict that involves a proposition belonging to a running action. If U is the set of *use* propositions on RsceAtt belonging to the running actions, for each $u \in U$, we compute the maximal cliques containing u in the Possible Intersection Graph using the algorithm described in Appendix 1. If none of these cliques is over-consuming, then the plan remains executable.

2. If the capacity has increased and some goals have been previously abandoned during replanning attempts, we exploit this opportunity to re-introduce the goals and replan. This strategy is basic, but allows to handle situations such as the following one. If the camera becomes out of order for a while (capacity decreased by 1), the plan repair process will fail and the replanning attempts will progressively abandon all

goals except maybe the communication with the ground station (the robot can not take pictures, nor safely move). If the breakdown is recovered (capacity increased by 1), the replanning process will find a plan achieving all abandoned goals whose deadline is coherent with the current time.

However, this strategy should be improved. It would be useful for instance to be able to filter the abandoned goals and consider only the ones that could benefit from a resource capacity increase.

Action report

A report is associated with the ending timepoint of the corresponding action (et^a). An action either terminates by itself or is interrupted by an END command. In the latter case ($et^a \in WaitingEndTPs$), the action is removed from the set of running actions and the ending timepoint is mapped to its occurrence time only when the termination report is received.

In both cases, the reception time of the report (st^{cycle}) can either be consistent with the temporal network (and et^a is instantiated) or outside the bounds of the execution interval of et^a. In the latter case, the instantiation of et^a will lead to an inconsistent temporal network, unless we can replace it by a new timepoint without invalidating the temporal constraints of the plan.

If the message is received too late, two situations need to be considered, depending on whether there is some temporal flexibility left in the plan or not. As stated before[9], the only constraints that can be posted between et^a and another timepoint apart from the action a are flexible precedence constraints $]0, +\infty[$. To check the remaining flexibility, we consider all timepoints that are preceded by et^a. If there is some timepoint T for which the precedence link with et^a has been so squeezed that there is less than one timestep left between now and T[10], then the plan is not flexible enough to insert the new timepoint and guarantee the plan executability. A replanning process, with the immediate abandon of one goal is requested. On the contrary, if there is some flexibility, a new ending timepoint $et^{a\prime}$, set to st^{cycle}, is created and the failed timepoint et^a is relaxed.

If the message is received too early (typically, a failed action can terminate very soon), a new ending timepoint $et^{a\prime}$, set to t_{occ}, is also created, and the timepoints of the action occurring after st^{cycle} are considered to be failed and relaxed.

The new timepoint is constrained to occur before the future timepoints (i.e. a precedence constraint between $et^{a\prime}$ and each executable timepoint is added to the STN).

The relaxation of a failed timepoint amounts to removing the temporal constraints on the timepoint from the set of constraints and recomputing the network. These operations keep the temporal network consistent.

[9] cf. page 103.

[10] $(T - et^a)$ in $]0, timestep + st^{cycle} - et^a_{ub}]$

```
INTEGRATING_REPORT()

% Report($et^a$,$status$,$logState$,$rsceLevels$)
% $et^a$: the report corresponds to the action $a$ and is associated with its ending timepoint $et^a$
% $status$: ending status of the action (nominal, failed, interrupted)
% $logState$: if the report is not nominal, $logState$ contains the value returned by the
%             controlled system for each grounded logical attribute relevant to $a$
% $rsceLevels$: set containing the level at the end of $a$ returned by the controlled system
%               for each allocated resource used by the action
% $flawAttributes$: list of attributes invalidated by the report (open conditions, resource conflicts)
% $lastFlawTP$: starting timepoint of the last flaw revealed in the plan

$newRepairSearchTree \leftarrow$ true
possibly remove $et^a$ from $ExecutableTPs$, $ExecTPs$, $WaitingEndTPs$
remove $a$ from $RunningActions$
if (($et^a$ is executable) or ($et^a \in WaitingEndTPs$)) and ($st^{cycle} \in [et^a_{lb}, et^a_{ub}]$)
    % message received inside the execution interval of $et^a$
    set occurrence time: $((et^a - st^H) = st^{cycle})$
else if ($st^{cycle} > et^a_{ub}$) % time-out
    $noFlexTimeout \leftarrow$ check the temporal flexibility of the plan
    if ($noFlexTimeout$)
        $abort \leftarrow$ true
        $repair \leftarrow$ false
    else
        create new timepoint $et^{a\prime}$
else
    create new timepoint $et^{a\prime}$
$needRepair \leftarrow$ false
if ($logState$ not empty)
    ($needRepair$, $RunningActionsWithThreat$) $\leftarrow$ insert new state
if ($rsceLevels$ not empty)
    for each allocated resource Rsce relevant to $a$
        compute its theoretical level
        ($needRepair$, $abort$) $\leftarrow$ check and adjust the level
        if ($needRepair$)
            $rsceAttributes \leftarrow rsceAttributes \cup \{$Rsce$\}$
if (!$abort$) and ($needRepair$)
    ($flawAttributes$,$lastFlawTP$)$\leftarrow$ get open conditions revealed in $ExecutingPlan$
    if ($rsceAttributes$ not empty)
        $flawAttributes \leftarrow flawAttributes \cup rsceAttributes$
        $lastFlawTP \leftarrow et^H$
    REMOVE_CAUSAL_LINKS($flawAttributes$,$lastFlawTP$)
    $repair \leftarrow$ true
forget the past
update $ExecutableTPs$ and $ExecTPs$
```

Table 3.4: *Integration of a "report" message*

If the report contains information about the state, the logical and resource states of the system are updated in the following way:

State variables - The logical state of the system $LgcS_t$ contains the last value for each grounded logical attribute (or similarly the last executed event). If the report is nominal, $LgcS_t$ is updated with the effects of the action expected in the plan. Otherwise, it is updated with the values returned in the report ($logState$).

In IxTeT-eXeC, the value of a grounded attribute $latt_g$ is updated to the new value $v \in logState$ through the insertion of an explained event $event(latt_g : (*, v), t)$ occurring at $t = et^a$ or $t = et^{a'}$ according to the situation. Such an event may or may not be inserted in the plan. It is not inserted if it leads to a non executable plan, i.e. threatens some proposition of a running action a_r. In that case, and if a_r is preemptive, its interruption is requested ($a_r \rightarrow RunningActionsWithThreat$). Otherwise, the event is inserted. If the new value is in contradiction with some propositions of the failed action, or with some causal links, the causal links are broken and the propositions are removed. Thus, the sate update may reveal new open conditions to re-establish ($needRepair = true$).

Consider for instance the plan in Figure 3.2 and the execution situation illustrated in Figure 3.6. If a failure occurs during the execution of the first move action (e.g failure of map building) and stops the robot in an intermediate location (5,1), the procedural executive returns: $logState$ = {(PTU_POSITION(), forward), (MVT_GEN_INIT(), false), (ROBOT_STATUS(), still), (AT_ROBOT_X(), 5), (AT_ROBOT_Y(), 1)}. This report is received too early according to the execution interval of the timepoint t_{16} which is relaxed, whereas the timepoint t_{21} is created. The insertion of the new state breaks causal links on the attributes MVT_GEN_INIT(), AT_ROBOT_X() and AT_ROBOT_Y(), thus revealing 5 open conditions to re-establish.

Resource level - For each resource r relevant to the action, the report returns the current "real" level l_r. The "theoretical" level of the resource in the partial plan $RscL_t(r)$ is computed according to the formula presented in section 3.2.2. $RscL_t(r)$ is then compared to l_r and adjusted according to the schema in Figure 3.7 (cases (a) and (b) refer to the ones presented in section 3.2.2).

- If $l_r \in [RscL_{lb}, RscL_{ub}]$, the plan is consistent with reality. If the exact level can be computed (case (a)), its value is updated by adding and propagating the constraint $RscL_t(r) = l_r$.
- If ($l_r < RscL_{lb}$), the over-consumption is reflected on the plan by adding a consumption[11] of quantity $c = RscL_{lb} - l_r$. In case (a), the new level ($RscL'_t(r) = RscL_t(r) - c$) is updated by adding the constraint $RscL'_t(r) = l_r$, or equivalently $RscL_t(r) = RscL_{lb}$. If some running action modifies r (case (b)), the part of the level due to past actions ($L_t^{past}(r)$) is updated by considering the worst case and adding the constraint $Lev_t^{past}(r) = lev_{lb}^{past}$.

[11] The mirror production of the complementary resource is also added if r is a reservoir resource.

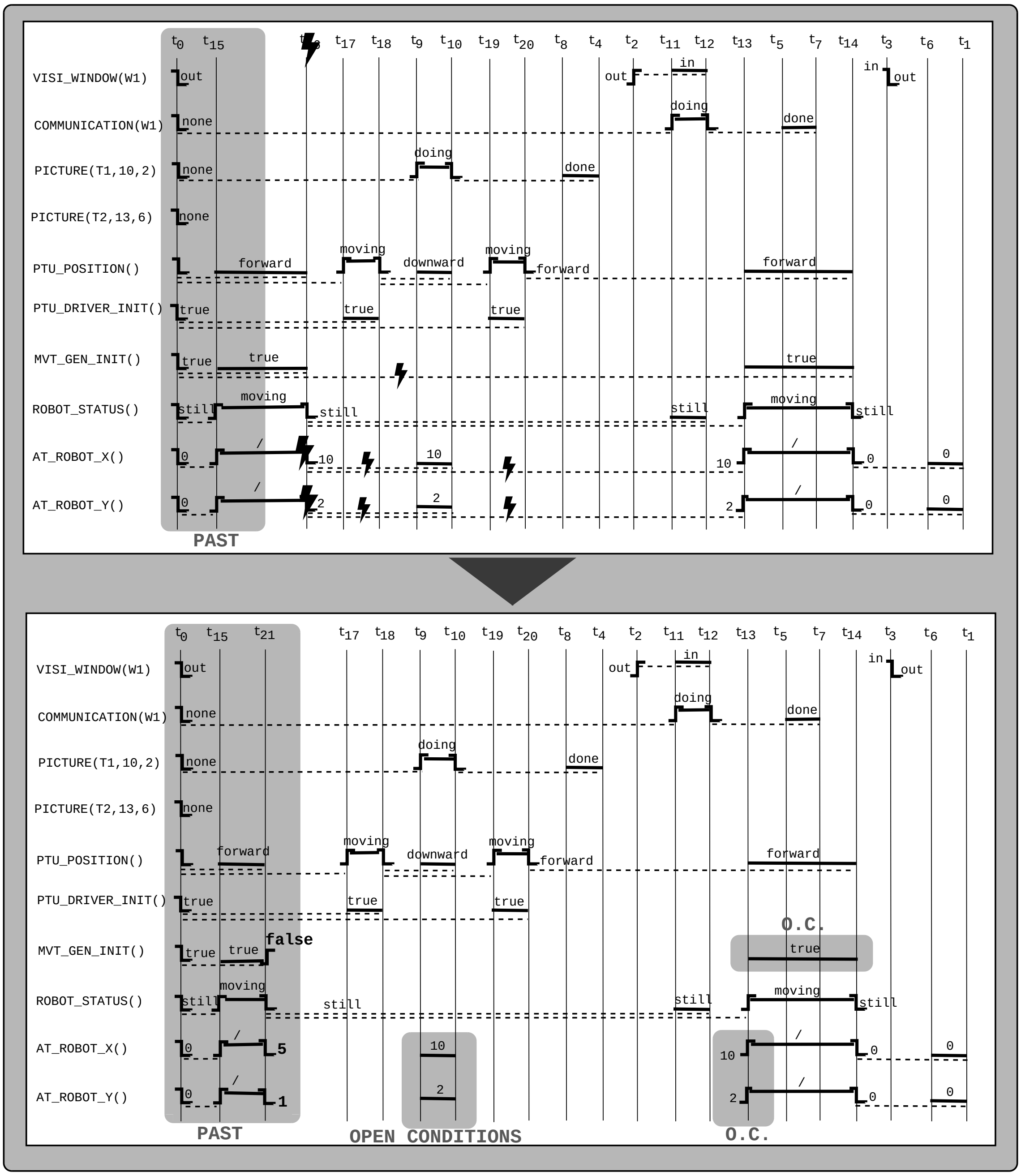


Figure 3.6: *Failure report insertion*

- If ($l_r > RscL_{ub}$), the over-production is reflected on the plan by adding a production[12] of quantity $p = l_r - RscL_{ub}$. Similarly, the level is updated by adding the constraint $RscL_t(r) = RscL_{ub}$ in case (a), and $Lev_t^{past}(r) = lev_{ub}^{past}$ in case (b).

[12] The mirror consumption of the complementary resource is also added if r is a reservoir resource.

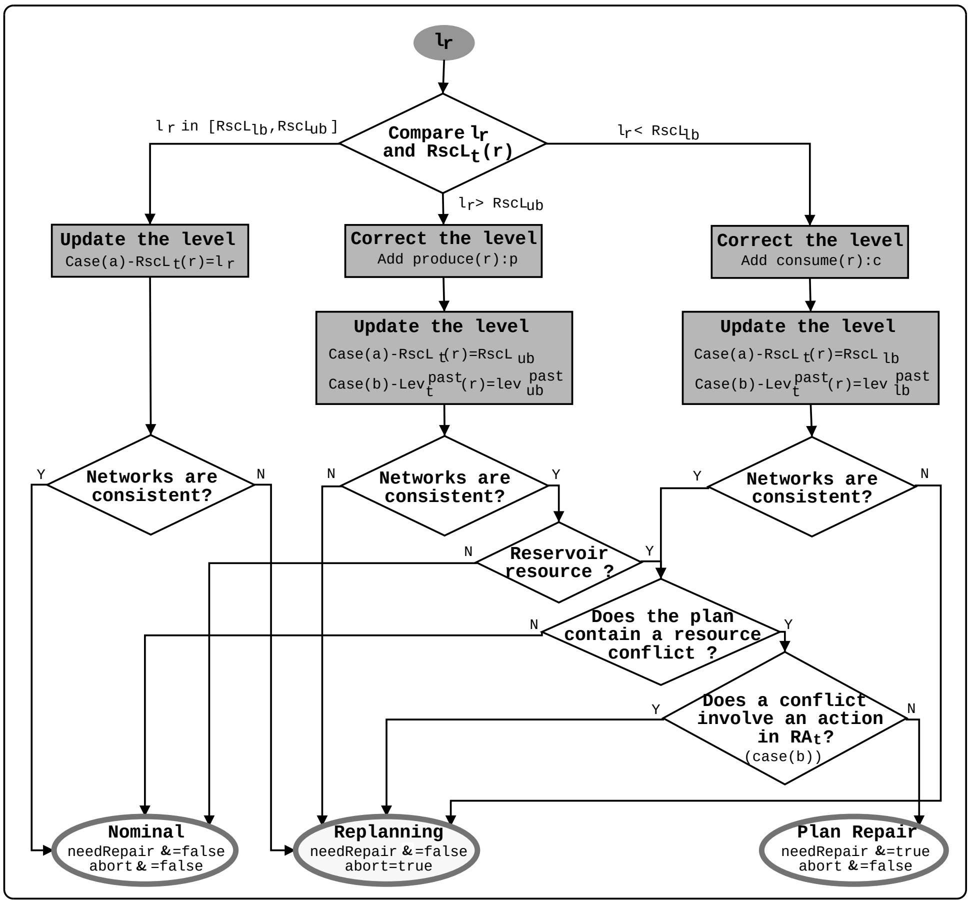


Figure 3.7: *Check and adjust the resource level*

In case of over-consumption or in case of over-production of a reservoir resource, a conflict may appear in the plan and require a plan repair. The existence of a conflict is checked using the maximal cliques algorithms. In case (b), it is also necessary to check that no conflict concerns a running action. Depending on the plan executability, a plan repair or a replanning process is requested.

In conclusion, the report integration leads either to: a nominal plan whose execution can be pursued, to a non executable plan requiring the execution abortion, or to an executable plan with open conditions and/or resource conflicts requiring a plan repair. In the last case, we also need to break additional causal links: either between et^{cycle} and the starting timepoint of the last open condition, or between et^{cycle} and et^{H} if the plan contains a resource conflict.

3.4.2 Deletion of additional causal links

```
REMOVE_CAUSAL_LINKS($Att^{flaw}$, $lastTP$)

% $Att^{flaw}$: set of grounded attributes involved in the flaws (resource conflicts and/or open conditions)
%           present in the current partial plan
% $lastTP$: timepoint corresponding to the starting timepoint of the latest open condition if the plan
%           contains no resource conflict, to $et^H$ otherwise
% $LogAtt^{insert}/RsceAtt^{insert}$: set of grounded or partially grounded logical/resource attributes appearing
%           in the (partially grounded) models of actions possibly inserted to repair the flaws
% $Event^i$: set of events on the attributes in $LogAtt^{insert}$ occurring before $lastTP$
% $Assertions(e)$: set of hold propositions established by the event $e$ and imposed by an action model
% $CausalLinks(e)$: set of hold propositions established by the event $e$ and corresponding to causal links
% $Extension(e)$: interval during which the value established by the event $e$ is imposed by some assertions

% 1- Compute $LogAtt^{insert}$
$LogAtt^{insert} \leftarrow \emptyset$
$RsceAtt^{insert} \leftarrow \emptyset$
$NewAtt^{insert} \leftarrow Att^{flaw}$
while ($NewAtt^{insert}$ not empty)
    select and remove $AttName_i(param_i)$ from $NewAtt^{insert}$
    if ($AttName_i$ is logical attribute)
        $LogAtt^{insert} \leftarrow LogAtt^{insert} \cup \{AttName_i(param_i)\}$
    else
        $RsceAtt^{insert} \leftarrow RsceAtt^{insert} \cup \{AttName_i(param_i)\}$
    $ActionModels(AttName_i(param_i)) \leftarrow$ get partially grounded models of actions for
                                          which $AttName_i(param_i)$ is a main attribute
    for each action model $A_g \in ActionModels(AttName_i(param_i))$
        for each $AttName(param) \in A_g$
            if ($AttName(param) \notin (NewAtt^{insert} \cup LogAtt^{insert} \cup RsceAtt^{insert})$)
                $NewAtt^{insert} \leftarrow NewAtt^{insert} \cup \{AttName(param)\}$

% 2- Compute the causality chains and select causal links to break
compute $Event^i$
for each event $e$ in $Event^i$
    compute $Assertions(e)$, $CausalLinks(e)$ and $Extension(e)$
    $RemovableCausalLinks(e) \leftarrow$ get c.l. that do not belong to $Extension(e)$ and
                                       do not support a running action
    $CausalLinksToRemove \leftarrow CausalLinksToRemove \cup RemovableCausalLinks(e)$

% 3- Break causal links
delete propositions $\in CausalLinksToRemove$
```

Table 3.5: *Removal of additional causal links*

After the integration of a message, the plan may contain flaws on a set of grounded attributes Att^{flaw}. Fixing these flaws may require the insertion of new actions. Let us note $LogAtt^{insert}$ and $RsceAtt^{insert}$ the sets of logical and resource attributes appearing in these actions. Additional causal links, protecting the establishment of propositions on attributes in $LogAtt^{insert}$, have to be broken to allow the insertion of new actions in the current structure of the plan.

Consider for instance the goal integration example given in Figure 3.5 (page 113). The establishment of the open condition *hold*(PICTURE(T2,13,6): *done*) requires the insertion of the action Take-image(T2,13,6). This insertion may introduce new open conditions on the attributes PTU_POSITION(), AT_ROBOT_X(), AT_ROBOT_Y() which in turn may require the insertion of Move and MovePTU actions, etc.

The determination of the set $LogAtt^{insert}$ is based on information given by the abstraction graph generated offline from the model description (see section 2.3.4). This abstraction graph notably records which attributes are linked to the primary effects of each action. The primary effects are used to select an action in order to establish an open condition or to produce a resource. These attributes, which justify the insertion of an action, are also called **main attributes**.

Starting from the grounded attributes in Att^{flaw}, $LogAtt^{insert}$ and $RsceAtt^{insert}$ are recursively computed by considering the partially grounded models of actions[13] for which the attributes in $LogAtt^{insert}$ and $RsceAtt^{insert}$ are main attributes, and adding the (grounded or not grounded) attributes contained by the actions and not yet taken into account in $LogAtt^{insert}$ and $RsceAtt^{insert}$; and thus until a fixed point is reached.

Causal links on the attributes in $LogAtt^{insert}$ are removed, but not all of them. The selection of the causal links to break is based on the work presented in [Gaborit 96].

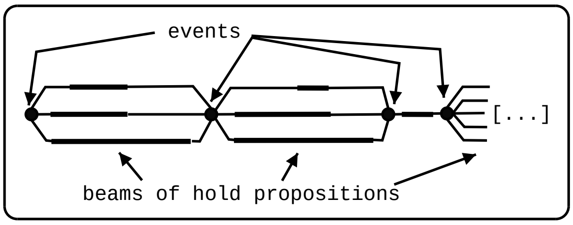


Figure 3.8: *Chain of causality for an attribute*

In a solution plan, the events on a same attribute are totally ordered and form a chain of causality. The *hold* propositions that are established by the same *event e* form a “beam” which can be separated into two sets:

- *Assertions*(e) contains the *hold* propositions imposed by the initial scenario or by

[13] Consider the action model A, for which $AttName_i$ is a main attribute, and $AttName_i(param_i) \in LogAtt^{insert}$. A has several potential partial instantiations. The set of these partially grounded models of actions are computed by considering successively each $event(AttName_i(param_{event}))$ and binding $param_{event}$ and $param_i$. Similarly, if $AttName_i(param_i) \in RsceAtt^{insert}$, the partially grounded models of actions are computed by considering the *produce* propositions.

an action in the plan,

- $CausalLinks(e)$ contains the *hold* propositions that have been inserted during the planning process to serve as causal links.

Figure 3.8 represents such a chain of causality. In IxTeT-eXeC, the breaking process is applied to the causality chains of the attributes in $LogAtt^{insert}$ formed by the events occurring before $lastTP$ ($Event^i$).

Among the causal links of a beam, some do not need to be broken. P. Gaborit defines the **extension** of an event e as the set of assertions that necessarily extend the value established by e. If $e = event(AttName_i(param_i) : (*, *), t)$ and $h = hold(AttName_i(param_i) : *, (t_1, t_2))$ then $Extension(e)$ is recursively defined by the equations:

$$\begin{array}{rcl} h \in Extension(e) & \Leftarrow & ((h \in Assertions(e)) \text{ and } (t_1 = t)) \\ h \in Extension(e) & \Leftarrow & ((h \in Assertions(e)) \text{ and} \\ & & (\exists h' = hold(AttName_i(param_i) : *, (t'_1, t'_2)) \text{ such that} \\ & & (h' \in Extension(e)) \text{ and } (t_1 \leq t'_2))) \end{array}$$

The first equation encompasses the assertions that intersect necessarily the event, whereas the second equation adds to $Extension(e)$ the assertions that necessarily overlap a proposition in $Extension(e)$.

Therefore, a causal link $cl = hold(AttName_i(param_i) : *, (t_1, t_2)) \in CausalLinks(e)$ can be removed if cl is not necessarily covered with assertions that belong to $Extension(e)$, i.e.

$$\nexists h = hold(AttName_i(param_i) : *, (t'_1, t'_2)) \text{ such that } (h \in Extension(e)) \text{ and } (t_2 \leq t'_2)$$

Figure 3.9 illustrates this deletion criterion.

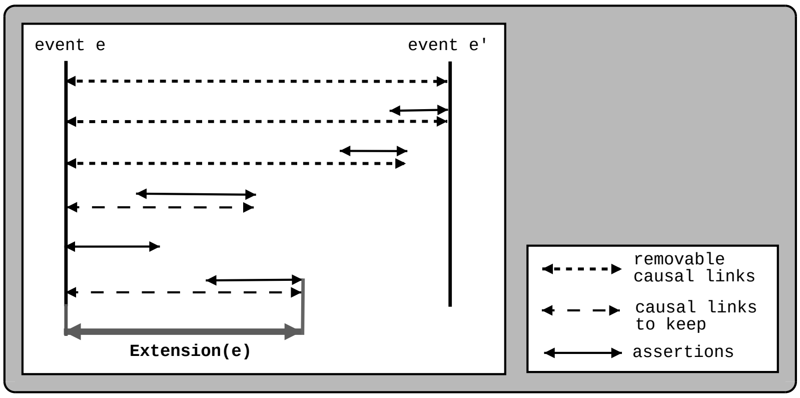


Figure 3.9: *Application of the deletion criterion*

Finally, a causal link is removed if it satisfies the above condition and if it does not take part in the establishment of a proposition that belongs to a running action.

At the end of this process, the plan is executable, the causality chains of a subset of attributes are broken, but the set of actions independent from the failures and their repair remain executable.

3.5 Plan repair

```
Plan_Repair()

% $\mu$: percentage that corresponds to the share of a timestep allocated to the sensing and plan repair
%     phases of the cycle
% $t_{limit}$: deadline for planning in the current cycle, $t_{limit} = st^{cycle} + \frac{\mu}{100} * timestep$

get $t_{limit}$
if ($newRepairSearchTree$)
    set $ExecutingPlan$ as root of the new search tree
$newRepairSearchTree \leftarrow$ false
$SolutionFound \leftarrow$ false
$NoSolution \leftarrow$ false
while (!$SolutionFound$) and (!$NoSolution$) and (current_time $< t_{limit}$)
    ($SolutionFound$, $NoSolution$) $\leftarrow$ plan_one_step($et^{cycle}$, $RunningActions$)
if ($NoSolution$)
    $abort \leftarrow$ true
    $repair \leftarrow$ false
if ($SolutionFound$)
    $ExecutingPlan \leftarrow$ solution plan
    $repair \leftarrow$ false
else
    $ExecutingPlan \leftarrow$ get last executable partial plan
update $ExecutableTPs$ and $ExecTPs$
```

Table 3.6: *Plan Repair during one cycle*

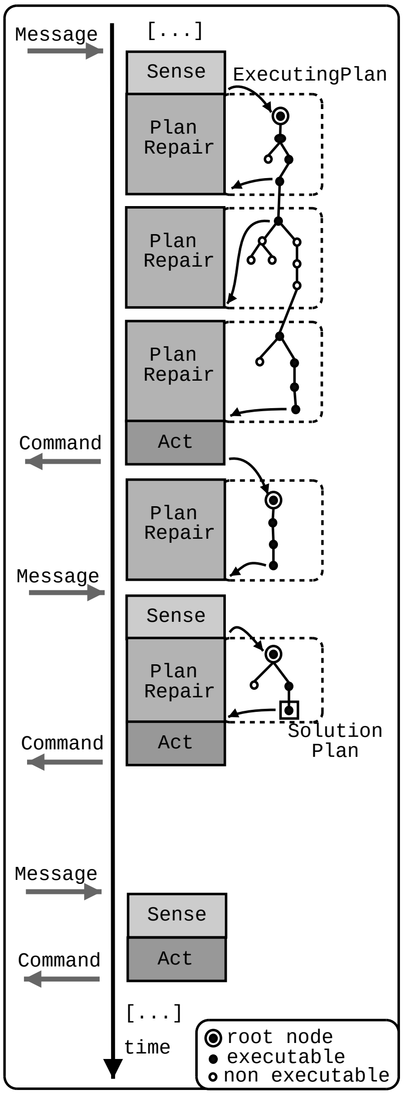


Figure 3.10: *Interleaved planning and execution*

The plan repair process is similar to the IxTeT search process in the plan space, but distributed (if necessary) on several cycles, thus allowing the executive to be responsive during the intermediate "action" and "sensing" phases (cf. Figure 3.10).

Distributing planning on several cycles raises two important problems:

1. **On which plan relies the execution in the "act" phase of a cycle, especially if no solution has been found?** This plan has to be *executable*. At each planning step, the node is labeled if the corresponding partial plan fully supports the running actions. When the maximum time allowed to plan repair in the cycle is reached, and if the current partial plan is not acceptable, the last executable node is chosen and its corresponding plan becomes *ExecutingPlan*.
2. **On which plan and which search tree relies the planning process in the next cycle?** If no decision has been made meanwhile (no timepoint execution, no message reception), the search tree can be kept as is and further developed during the next "plan repair" phase. It is even possible to backtrack on decisions made in previous cycles. However, if the plan has been modified, a new search tree is mandatory ($newRepairSearchTree$ = true). Its root node is made of the new *ExecutingPlan*. The planning decisions made in the previous cycles are now fixed, no backtrack is possible.

During the "plan repair" phase of each cycle, planning is done one step at a time until it results in a dead-end ($NoSolution$ = true), or a solution is found ($SolutionFound$ = true), or a deadline (t_{limit}) is reached. This deadline corresponds to the percentage (μ) of the timestep that is dedicated to the message integration and the plan repair phases. The parameter μ can be tuned by the user.

Slight modifications have been made to the IxTeT planning process.

- In order to take current time and planning duration into account, each time a new timepoint T is inserted in the plan, it is constrained to occur after the end of the current cycle ($T > et^{cycle}$).

- The flaw analysis process now records the identifiers of the actions involved in each flaw.
- A step has been added in the development of a search node. If the insertion of the resolver leads to a (locally) consistent plan, the support status of the running actions is checked. More precisely, we check that they are not involved in the flaws found by the analysis module, as well as in the flaws on the lower levels of abstraction. For that, we consider, for each running action, its logical propositions on attributes belonging to lower abstraction classes and check that they are established and not threatened. Likewise, we consider the resource propositions and check that they are not contained by an over-consuming maximal clique. A node is labeled if the running actions are supported.

At the end of the plan repair phase of a cycle, we search for the last labeled node, build the corresponding partial plan and check the global consistency of its networks. *ExecutingPlan* is updated with the last labeled and consistent plan.

We also implemented some mechanisms during the execution process that allow to reduce the space analysis in case of plan search process. These mechanisms are denoted "forget the past" in the algorithms presented in Table 3.4 and Figure 3.11. Each time a timepoint is executed, the sets of logical and resource propositions are updated. More precisely, it is of no use to look for threats or establishing events in the past. Thus, the past *hold* and *event* propositions are removed, except the last executed event for each grounded attribute. Similarly, the past *use* propositions are removed[14]. The *use* propositions which use a constant quantity of a resource, whose starting timepoint is in the past and which have the same ending timepoint in the future are aggregated in a unique proposition[15]. This aggregation mechanism may considerably reduce the size of the cliques in the PIG and thus the time needed for resource conflict detection. Consider for instance a plan for the rover domain with 6 image goals. It contains 6 Take-image actions, i.e. 6 *consume* propositions on the resource STORAGE (i.e. *use* propositions ending at et^H). After the execution of the second Take-image, the two first propositions are aggregated. The number of propositions potentially involved in a future conflict is further reduced to 4 after three Take-image executed actions, etc.

[14] Still, the information on the past productions and capacity increments are kept.

[15] Its quantity is the sum of the quantities used by the aggregated propositions.

3.6 Action

```
ACT()

$endCycle \leftarrow$ false
$WaitingExecTPs \leftarrow \emptyset$
while ($ExecTPs$ not empty) and ($!abort$) and ($!endCycle$)
   $(ExecutedTP, t_{exec}) \leftarrow$ get first timepoint in $ExecTPs$ to execute
   if ($t_{exec} > et^{cycle}$)
      % $ExecTPs$ can contain timepoints to execute in another cycle (wake up for Msg or repair)
      $endCycle \leftarrow$ true
   else
      execute $ExecutedTP$
      update $ExecutableTPs$, $ExecTPs$
if ($!abort$)
   add $WaitingExecTPs$ to $ExecTPs$
   send commands to stop the actions in $RunningActionsWithThreat$
else
   send commands to stop the actions in $RunningActions$ which are preemptive
```

Table 3.7: *Action*

Each timepoint t is associated with its *execution time* t_{exec}. Its value varies with the type of timepoint. In case of goal timepoints, starting timepoints of actions and ending timepoints of early-preemptive actions, $t_{exec} = t_{lb}$. In case of ending timepoints of late-preemptive actions, $t_{exec} = t_{ub} - timestep$. Finally, in case of ending timepoints of non-preemptive actions, $t_{exec} = t_{ub}$.

During the "action" phase, TEXEC successively executes the timepoints of the set $ExecTPs$, that contains the timepoints which are executable and whose execution time happens before the end of the current cycle. $ExecTPs$ is updated after each timepoint execution to take into account newly executable timepoints. The execution strategy also varies according to the type of the timepoint. It is summarized in Figure 3.11.

During a plan repair process, if a goal is not achievable or an action not executable, and if their execution interval has some flexibility left, their execution is postponed (→ $WaitingExecTPs$), and will be reconsidered in the following cycle. However, if no delay is admissible, a timeout situation occurs and the execution abortion and replanning process are requested. If the goal/action are achievable/executable, the timepoint is executed, i.e. it is instantiated and its value is propagated through the temporal network. On the contrary, the ending timepoint of a preemptive action is not instantiated when the command is sent, but upon the report reception (expected in the next cycle). Finally, TEXEC detects when the report associated with a non-preemptive action has not been received in time (before et^{a}_{ub}) and reports a timeout situation. However, TEXEC keeps waiting for the report until there is no flexibility left in the plan.

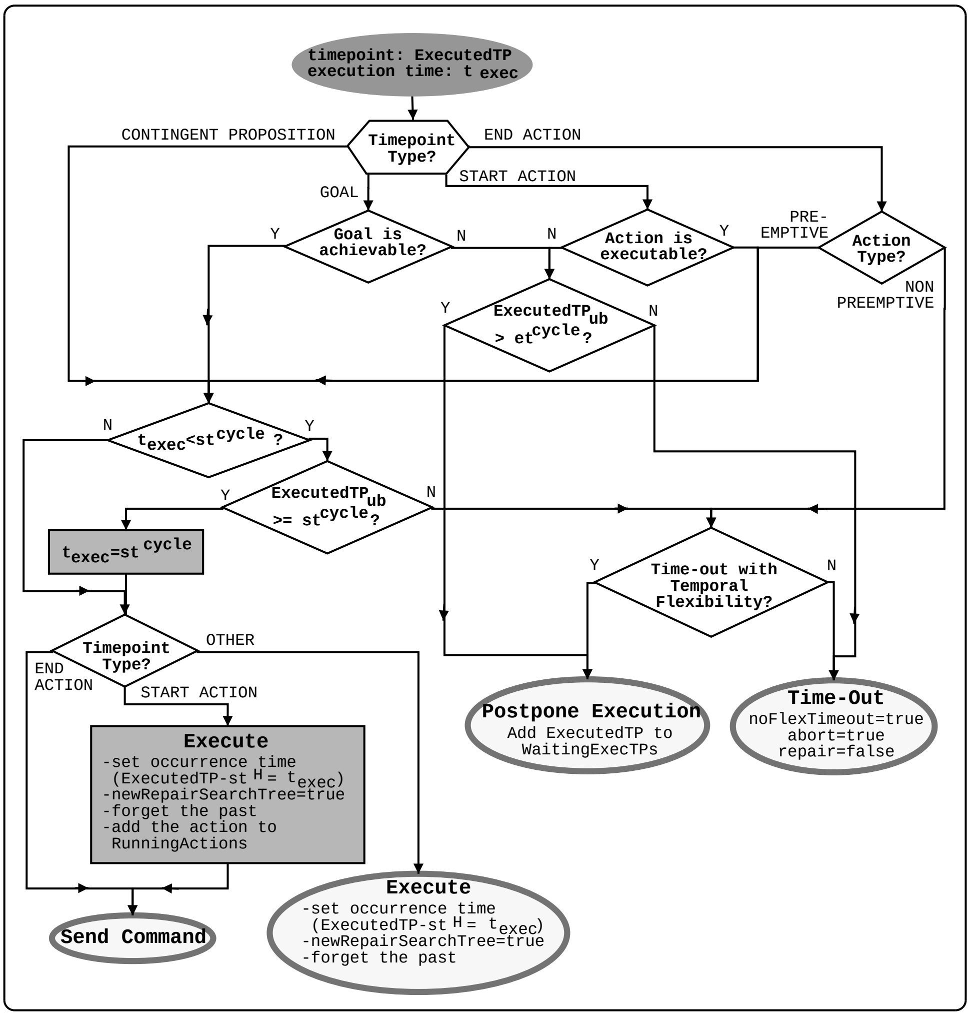


Figure 3.11: *Execution of a timepoint*

During a plan repair process, a record of the actions involved in the flaws detected during the analysis process, as well as the set of attributes considered during this analysis is kept along with *ExecutingPlan*. Thus, to check that an action is executable (similarly a goal is achievable) during the "action" phase, it is sufficient to check that the action is not contained by this set of actions and to analyze the flaws on the propositions involving non already analyzed attributes.

3.7 Complete replanning

If an execution abortion has been requested, cycling is stopped once the reports of all running actions have been received and integrated, leading to the partial plan $P_{t_s} = (\emptyset, FA_{t_s}, S_{t_s}, G_{t_s}, Co_{t_s}, C_{t_s}, L_{t_s}, F_{t_s})$. The planning problem is extracted from P_{t_s}. It corresponds to an initial plan $P_{init}^{attempt}$ that contains the state of the system at t_s, the future contingent profiles and the set of not yet completely achieved goals whose temporal constraints are coherent with current time. More precisely, a goal is rejected if its associated constraints (Dur, Orig and MinAchiev, cf. section 3.1.2) are incompatible with the time left between now and the planning horizon.

$$P_{init}^{attempt} = (\emptyset, \emptyset, S_{init}^{attempt}, G_{init}^{attempt}, Co_{init}^{attempt}, C_{init}^{attempt}, \emptyset, F_{init}^{attempt})$$

$$\begin{array}{rcl} \text{with} & & \\ S_{init}^{attempt} & = & S_{t_s}, \\ G_{init}^{attempt} & = & \{g \in G_{t_s} / \text{ temporal constraints on } g \text{ are coherent with current time}\}, \\ Co_{init}^{attempt} & = & \{\text{proposition } p \in Co_{t_s}/p \text{ occurs in the future}\}, \\ C_{init}^{attempt} & = & \{c \in C_{t_s}/c \text{ is a constraint just on variables appearing in } S_{init}^{attempt}, \\ & & Co_{init}^{attempt} \text{ and } G_{init}^{attempt}\} \\ & & (C_{init}^{attempt} \text{ notably contains constraints on origin and horizon timepoints}), \\ F_{init}^{attempt} & & \text{contains the open conditions in } G_{init}^{attempt}. \end{array}$$

The main difficulty in the replanning process comes from the uncertainty linked to the POCL planning technique. We do not know in advance how much time the planning process will take and we cannot interrupt it at any time and come up with an applicable plan. As illustrated in Figure 3.12, the problem is to guarantee that at the end of the replanning process, there remains enough time to execute the solution plan and meet the goal deadlines.

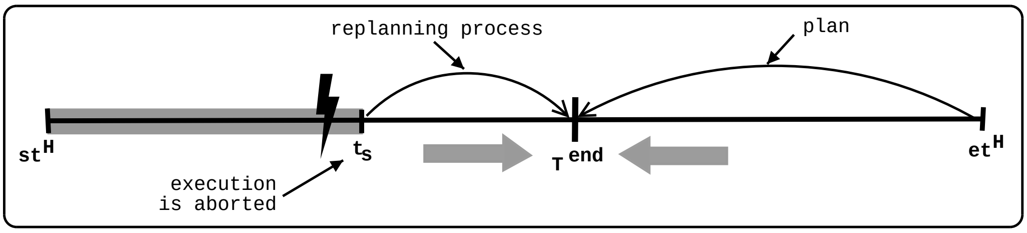


Figure 3.12: *Replanning and time progress*

We have been inspired by the approach used in the Remote Agent [Pell 97]: "planning to plan", that considers the planning process as one of the actions of the plan being elaborated. We propose to add a specific flexible timepoint T^{end} to the partial plan. This timepoint represents the end of the planning process. Initially, T^{end} is only constrained to occur between now and the end of the planning horizon. Its interval $[T_{lb}^{end}, T_{ub}^{end}]$ is further squeezed during the planning process: each time a new timepoint is inserted in the plan, it is constrained to happen after T^{end} (and before the end of the horizon). Thus T_{ub}^{end} decreases as new actions or new temporal constraints are added. The only way T_{ub}^{end} can increase is caused by a backtrack. During the planning process, there will not be enough time to execute the current partial plan if: $T_{ub}^{end} <$ current time.

The strategy consists then in planning one step at a time until it results in a dead-end ($NoSolution$= true), or a solution is found ($SolutionFound$= true), or a deadline $t_{replan-limit}$ is reached. $t_{replan-limit}$ is defined as $t_{replan-limit} = T_{ub}^{end} - d$, d being a *slack* duration (tuned by the user) to save enough time at the end of planning for execution cycle initialization. This deadline is updated after each planning step. Planning is stopped when $t_{replan-limit}$ is reached unless the next planning step corresponds to a backtrack node ($isBacktrackNode$= true). In that case, and if the next step increases $t_{replan-limit}$, planning is pursued.

If planning is aborted without finding a solution, a new planning attempt is done with fewer goals. A new initial plan $P_{init}^{attempt}$ is extracted from P_{t_s} as before. A first selection of the goals may reject the goals of the previous attempt whose temporal constraints are no more consistent with the current time. Otherwise one goal is selected. Is abandoned the goal with the lowest priority, and, if several goals have the same priority, the goal with the less flexibility for its achievement (this flexibility is computed as (st_{ub}^{g} - MinAchiev)). This criterion has been chosen to keep the goals that are more likely to be achieved in due course.

A drawback of this strategy is that the state of the controlled system is supposed to remain unchanged during the planning process. The solution found is valid with respect to this initial state. The advantage of our global approach is that, if the state has changed, the plan may be repaired once execution is started (resource updates, etc.). An improvement would consist in updating the system state (S_{t_s}) between each complete replanning attempt.

```
REPLANNING()

% $P_{init}^{attempt}$: initial plan for the replanning attempt
% $Goals$: set of goals to take into account in the replanning attempt
% $t_{replan-limit}$: deadline for the replanning attempt

$newPlanningAttempt \leftarrow$ true
$P_{t_s} \leftarrow ExecutingPlan$
$Goals \leftarrow \{g \in G_{t_s}/$temporal constraints on $g$ are coherent with current time$\}$
if ($noFlexTimeout$)
    select and remove one goal $g$ from $Goals$
    add $g$ to $AbandonedGoals$
    $noFlexTimeout \leftarrow$ false
else
    $Goals \leftarrow Goals \cup$
            $\{g \in AbandonedGoals/$temporal constraints on $g$ are coherent with current time$\}$
$P_{init}^{attempt} \leftarrow$ extract_initial_plan($P_{t_s}$, $Goals$, current_time)
while ($newPlanningAttempt$)
    $NoSolution \leftarrow$ false, $SolutionFound \leftarrow$ false, $isBacktrackNode \leftarrow$ false
    set $P_{init}^{attempt}$ as root of the new search tree
    get $t_{replan-limit}$
    while (!$NoSolution$) and
            (((!$SolutionFound$) and (current_time $< t_{replan-limit}$)) or ($isBasktrackNode$))
        ($SolutionFound$, $NoSolution$, $isBacktrackNode$) $\leftarrow$ plan one step
        update $t_{replan-limit}$
    if ($SolutionFound$) and (current_time $< t_{replan-limit}$)
        % A solution has been found in time
        $ExecutingPlan \leftarrow$ solution plan
        $newPlanningAttempt \leftarrow$ false
    else
        $Goals \leftarrow \{g \in G_{init}^{attempt}/$temporal constraints on $g$ are coherent with current time$\}$
        if ($Goals$ is empty) or (card($Goals$) == card($G_{init}^{attempt}$) == 1)
            $newPlanningAttempt \leftarrow$ false
            $ExecutingPlan \leftarrow$ extract_initial_plan($P_{t_s}$, $\emptyset$, current_time)
        else
            % New replanning attempt with fewer goals
            if (card($Goals$) == card($G_{init}^{attempt}$))
                select and remove one goal $g$ from $Goals$
                add $g$ to $AbandonedGoals$
            $P_{init}^{attempt} \leftarrow$ extract_initial_plan($P_{t_s}$, $Goals$, current_time)
% Cycle initialization
$replan \leftarrow$ false, $abort \leftarrow$ false
$st^{cycle} \leftarrow$ current_time
$et^{cycle} \leftarrow st^{cycle} + timestep$
update $ExecutableTPs$, $ExecTPs$
```

Table 3.8: *Replanning attempts*

3.8 Discussion

3.8.1 Plan repair vs replanning

The plan repair process aims to provide reactivity and avoid aborting execution and completely replanning at each failure. It is not guaranteed to find a valid plan, notably because backtrack nodes may be frozen by execution. The plan repair process keeps all actions of the initial plan, and thus cannot take into account serendipitous effects, the resulting plan may also contain useless actions.

However, the restricted invalidation of the plan previous to its repair limits the search space, so that a repaired plan may be found in a few cycles. In parallel, the actions that are independent from the failure and from its repair (their supporting causal links have not been removed) can be launched as scheduled in the initial plan. Actually, plan repair is especially efficient and useful for temporally flexible plans and plans with some parallelism (some sets of actions can be executed independently).

This mechanism is also efficient to compensate for inadequate models of the numeric effects of actions (duration, resource usage). For instance, it is almost impossible to have an accurate estimation of the time taken by a robot to go from a location (x_1, y_1) to a location (x_2, y_2) in an unknown and dynamic environment. In the IxTeT-eXeC model, move(x_1,y_1,x_2,y_2) is defined as a late preemptive action. If the robot takes longer than expected in the model (e.g. due to obstacle avoidance), the action is interrupted. The controlled system returns the intermediate location (x_i, y_i) and, if some temporal flexibility remains, a new move(x_i,y_i,x_2,y_2) is quickly inserted and launched. This example is simple but representative of the failures that frequently break plan execution.

When a plan adaptation is needed, the current strategy consists in trying plan repair while the plan remains *executable*, and then replanning if the plan repair fails (no solution exists) or times out. In some cases however, it could be useful to be able to detect that a plan repair is guaranteed to fail and avoid wasting time in a useless attempt. Such a situation happens for instance in case of capacity decrease of a non producible resource, when the future actions consume more than allowed by the new capacity. It can also happen when the temporal constraint of an open condition is too tight to allow the insertion of its establishing action(s). A future work direction could be the definition of a "repairable" plan criterion. Still, there may be a trade-off to consider between the time spent in the analysis of the plan structure and the time spent in plan repair cycles before the planning process detects by itself that no solution is possible, or times out.

3.8.2 Temporal controllability

In the first version of IxTeT-eXeC, the temporal network is encoded as a Simple Temporal Network defined by the set of timepoints present in the plan and the duration interval of the links between each pair of timepoints. In this context, all temporal links are considered equivalent, and the propagation algorithms may equally squeeze their duration intervals. However, these links have different roles in the plan.

Some correspond to precedence constraints and their initial interval $]0, +\infty[$ can actually be squeezed by the propagation of constraints, notably the plan horizon, thus reflecting the flexibility left in the plan.

Other links correspond to the duration of actions that are completely controlled by the system (a monitoring action for instance): the robot can interrupt the action without invalidating its nominal behavior and effects. In that case, the temporal link is considered *controllable* and can be squeezed by propagation without questioning the action execution success.

In many cases however, the actual duration of an action nominal execution cannot be controlled by the system. It notably concerns non preemptive actions, and some late preemptive ones (consider for instance a move action, which may take more or less time, depending on the obstacles along its path). The initial duration defined in the action model reflects the uncertainty on the real execution duration. If the interval of such a *contingent* link is tightened during propagation, the plan will not encompass all possible temporal scenarii, and the execution of this action is more likely to time out and require replanning (with no flexibility left in the plan).

Work is in progress (see [Gallien 04]) to increase the performances of IxTeT-eXeC by taking into account explicitly contingent links in the elaboration of plans.

This work is based on the STNU framework (*Simple Temporal Network with Uncertainty*) which makes a distinction between two types of links: controllable and contingent[16] ones. Several levels of controllability have been defined in this context [Vidal 99]. Two are of particular interest in our case:

1. **pseudo controllability -** A consistent temporal network is said *pseudo controllable* if none of its contingent links has been squeezed by the propagation process (e.g. a path consistency process PC1).
2. **dynamic controllability -** If a controllable timepoint T^{ctr} has strict constraints with respect to a contingent timepoint T^{ctg},[17] we want to guarantee that, whenever the corresponding contingent event occurs inside the bounds of T^{ctg}, these constraints are satisfied. In other words, we need a dynamic execution strategy, where the execution of each controllable timepoint is decided only according to the past observations, and a consistent execution is ensured under all possible scenarii. A consistent and pseudo controllable network is said *dynamically controllable* if such a dynamic execution strategy exists.

Dynamic controllability can be checked in polynomial time by the algorithm 3DC+

[16] In this framework, a contingent link has a duration interval $[l, b]$ such that $0 < l < b < \infty$.

[17] A contingent timepoint corresponds to the end timepoint of a contingent link, other timepoints are said controllable.

presented in [Morris 01, Huguet 02]. First, the consistency and pseudo controllability are checked by a path consistency algorithm. The 3DC+ algorithm then considers the relationships between each pair of controllable/contingent timepoints (T^{ctr}, T^{ctg}). The controllable timepoint may occur necessarily after T^{ctg} (case 1), necessarily before (case 2), or the timepoints are unordered (case 3). The first case does not raise any controllability issue. In case 2 and 3, the execution interval of T^{ctr} may be tightened to guarantee the dynamic controllability. In the last case, it may even be necessary to insert a so-called *wait* condition $< T^{ctg}, d >$ on the execution of T^{ctr}: wait for the execution of T^{ctg} or for at least a time delay d.

Let us consider the example of Figure 3.13. It represents a plan with two actions a_1 and a_2, both of contingent duration (respectively $[a, b]$ and $[c, d]$). a_2 is further constrained to occur during a_1. TEXEC will reason on the network of Figure 3.13b), which is consistent (and pseudo controllable) on the unique condition $b > c$. However, the dynamic controllability is ensured if the additional constraints $a > c$ and $b > d$ are satisfied (cf. Figure 3.13c)). In this example, the current execution strategy of TEXEC does not need to be modified and leads to the same execution scenario with the pseudo controllable network and the dynamically controllable one.

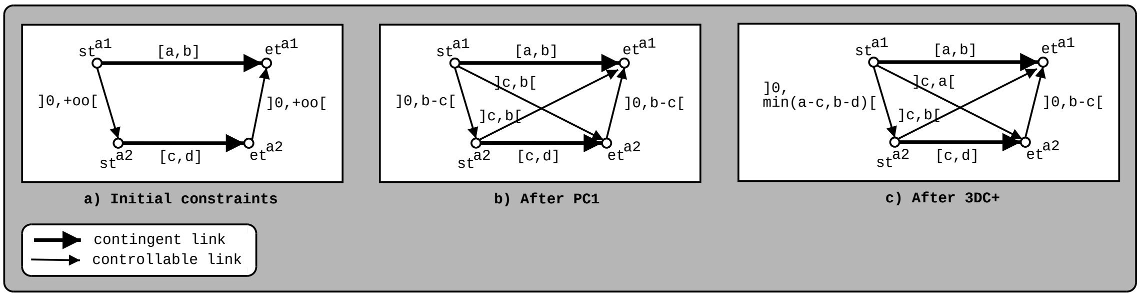


Figure 3.13: *Pseudo and Dynamic Controllability*

Consider now the example in Figure 3.14. The action a_2 is required to wait for an ending effect of action a_1 before being allowed to terminate. Depending on the respective durations of the contingent links ($b - c > a$ or $b - c < a$), a dynamic strategy may introduce a wait condition or not. We present in both cases the plan on which TEXEC currently relies ("after PC1"), and the same plan after the application of 3DC+. We detail the expected behavior of TEXEC in the four cases:

- In cases **(1)** and **(3)** (i.e. the current implementation, without 3DC+), st^{a_1} is executed at $t = 0$.[18] st^{a_2} is then executed at $t = 1$ and propagation restricts the bounds of et^{a_1} to $[5, 5]$ and et^{a_2} to $[4, 4]$. The network is no more pseudo controllable and the execution is likely to fail.
- In case **(2)**, st^{a_1} is similarly executed at $t = 0$. But st^{a_2} is only executed at $t = 4$,

[18] For clarity purpose, we omit the open bound. st^{a_1} is actually instantiated at $t = 0.001$.

and the contingent links are not squeezed. Thus, if the duration model is correct, the execution will not encounter temporal failures.

- In case **(4)**, the TeXeC execution strategy can not be used as is. We need to extend the definition of an *executable timepoint* (cf. section 3.2.3):
  A timepoint T is executable at time t if:
  1. *all timepoints T^p that must directly precede it in the temporal network have already been executed ($T^p_{lb} = T^p_{ub} = t^p < t$); and*
  2. *for each controllable link (T^p, T) that contains a wait condition on the contingent timepoint T^w: $< T^w, d >$, then T^w has already been executed or $t \geq t^p + d$; and*
  3. *t is comprised in the execution interval of T.*

  A priori, the definition of the execution time of timepoints does not need to be modified.

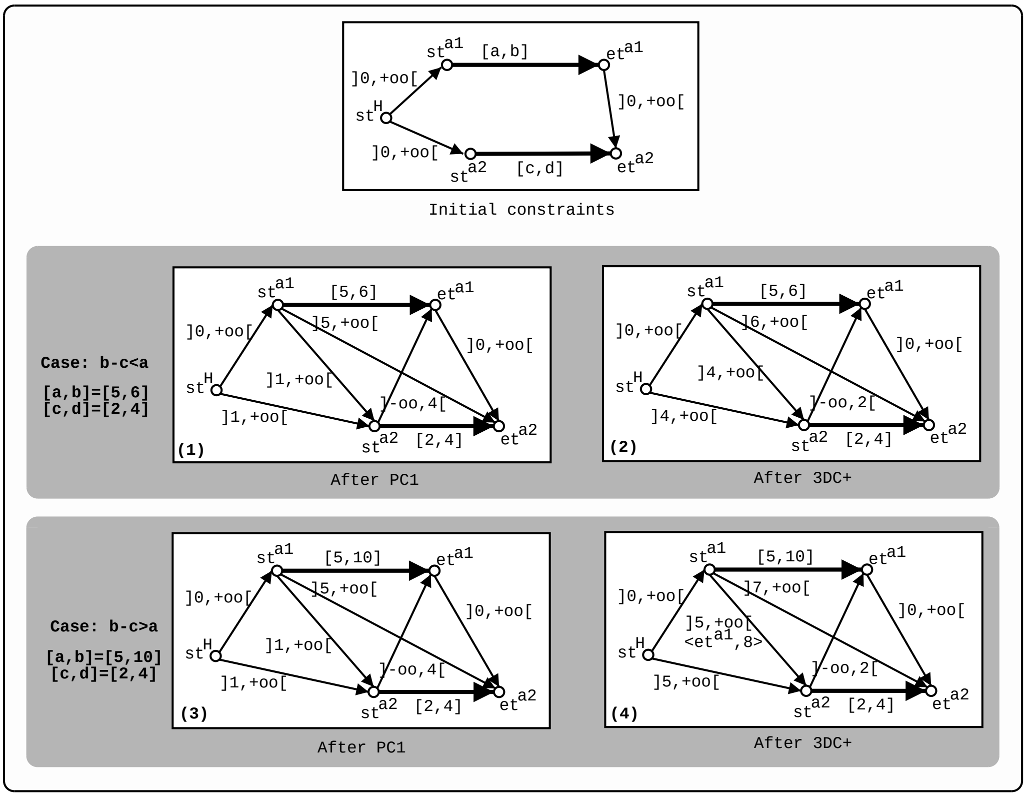


Figure 3.14: *Dynamic Controllability and Execution*

As we can see in these examples, the performances of IxTeT-eXeC could benefit from a checking of the dynamic controllability. Our system proposes mechanisms to repair temporal failures, but the number of possible failures can be reduced by imposing some controllability criterion. In fact, three criteria can be applied to the network: only consistent, consistent and pseudo controllable, consistent and dynamically controllable. They can be added to the definition of a *solution plan* and/or the definition of an *executable plan*. Future work should study which criterion needs to be chosen and in which conditions, considering that the 3DC+ algorithm will make the propagation process heavier, and that it implies a good confidence in the model of the contingent durations.

Another important issue needs to be considered: the impact on the atemporal network if some contingent temporal link (T^i, T^j) is involved in a mixed constraint $?x = c{*}(T^j - T^i)$. Likewise the atemporal variable $?x$ is not controllable but the propagation algorithms may reduce its domain, thus reducing the contingent duration interval. The differentiation between controllable and non controllable variables and its influence on the notion of consistency has been dealt with in [Fargier 96], but for discrete CSPs.

Chapitre 4

Les expérimentations menées au LAAS sur divers robots mobiles s'appuient sur une architecture logicielle. Dans ce chapitre, nous décrivons comment IxTeT-eXeC a été intégré dans l'architecture LAAS pour contrôler un robot mobile d'extérieur avec la mission d'exploration présentée au premier chapitre.

L'architecture LAAS

L'architecture LAAS se décompose en trois niveaux :

- Le *niveau fonctionnel* réalise tous les services et traitements de base associés aux capteurs/effecteurs. Ces services sont encapsulés dans des modules qui sont activés par des requêtes, retournent des bilans et exportent des données.
- Le *niveau de contrôle des requêtes* filtre les requêtes selon l'état courant du robot et un modèle formel des états permis et interdits.
- Le *niveau décisionnel* comprend tous les processus délibératifs : planification, reconnaissance de situation, etc.

La navigation autonome dans un environnement inconnu est un vaste domaine de recherche. Nous exploitons les algorithmes développés au LAAS pour : la modélisation de l'environnement (carte 3D), la localisation du robot (stéréo-odométrie) et la planification de trajectoires. Ces tâches sont réalisées par le niveau fonctionnel.

Le niveau décisionnel comprend IxTeT-eXeC et un exécutif procédural OpenPRS qui interagit avec l'opérateur, les modules fonctionnels et TeXeC. Si l'exécutif temporel décide du lancement et de l'arrêt des actions du plan, l'exécutif procédural de son côté :

- affine chaque action en requêtes pour les modules,
- surveille l'exécution de chaque action et calcule les bilans retournés à TeXeC,
- surveille les ressources,
- dispose d'une certaine latitude pour réagir rapidement à certains échecs,
- met le robot dans un mode d'attente lorsque une replanification est en cours.

Les modèles

Cette section détaille les modèles (IxTeT et OpenPRS) utilisés pendant l'expérimentation. Une partie des actions ont été réalisées par le robot, une autre partie a été simulée à bord ("communication avec une station-sol", "téléchargement des images").

Résultats

Nous présentons trois exécutions différentes pour un même scénario initial. Ils illustrent la capacité du système à réagir à des échecs temporels (obstacles ralentissant le robot) tout en respectant des échéances (fenêtres de visibilité pour les communications). Il illustrent également comment la flexibilité au niveau des ressources permet de traiter l'incertitude sur leur utilisation (taux de compression effectif des images connu au moment de l'exécution).

Chapter 4

Integration in the LAAS Architecture

One of the main reasons to extend IxTeT with execution, dynamic planning and replanning capabilities is to use it on board autonomous systems such as rovers or satellites. Those systems are usually developed and deployed using a particular architecture and its associated tools. We used the LAAS architecture to integrate IxTeT-eXeC.

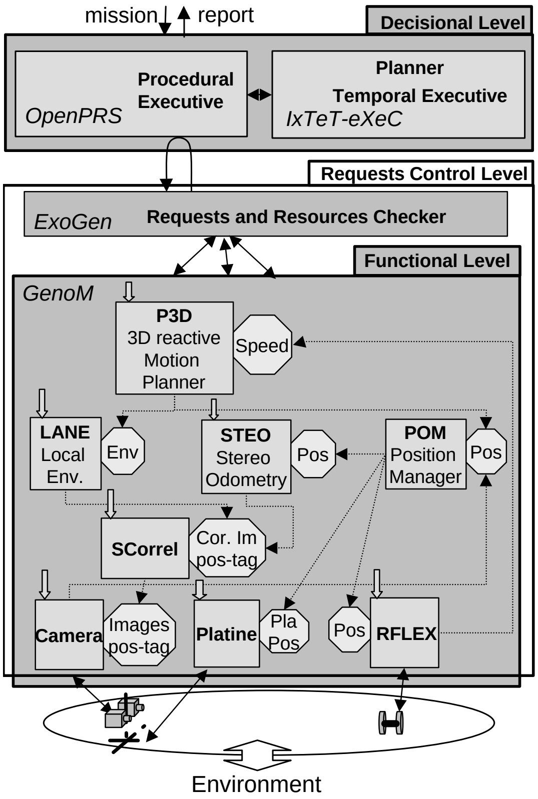


Figure 4.1: *Instantiation of the LAAS architecture on Dala*

The LAAS architecture [Alami 98] provides a support in order to design and integrate complex autonomous systems. This architecture has three levels, with different temporal constraints and uses different data representations. From the bottom to the top, the levels are:

- **The functional level-** It includes all the basic built-in robot action and perception capabilities, encapsulated into controllable communicating modules. Modules are activated by requests, send reports upon completion and export data.
- **The requests control level-** It filters the requests according to the current state of the system and a formal model of allowed and forbidden states.
- **The decisional level-** It includes the deliberative capabilities of the robot such as: producing actions plans, recognizing situations, fault detections, etc. IxTeT-eXeC has been integrated in this level and interacts with the procedural executive OpenPRS.

Figure 4.1 presents the architecture as set for the experiment[1] on Dala, an iRobot ATRV (cf. Figure 4.2). Dala runs a Pentium III (850 MHz) under Linux and is equipped with the following sensors: odometry, a stereo camera pair mounted on a pan&tilt unit (PTU) and a Sick laser range finder. IxTeT-eXeC has been deployed on Dala with the exploration mission scenario described in section 1.5.

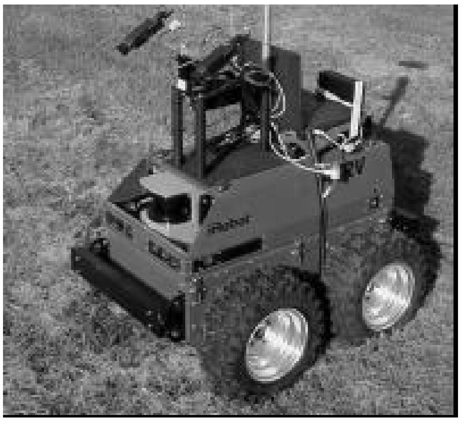


Figure 4.2: *The outdoor robot Dala*

The first section of this chapter gives a short description of the concepts and tools on which each level relies, and further details the role of the procedural executive in this context. The next section presents how the experiment has been set up, especially the models developed for the decisional components. In the last section, we consider how the robot, controlled by IxTeT-eXeC, did perform a specific mission on different runs.

[1] In fact, the Requests and Resource Checker has not been used during our tests.

4.1 The LAAS architecture

4.1.1 Functional level

The functional level is made of a set of controllable communicating modules. Each module encapsulates operational functions that can be activated, interrupted or parameterized by asynchronous *requests* sent to the module. Upon completion or abnormal termination, reports (with status) are sent back to the requester, which can be the procedural executive as well as another module or an operator. During the execution of functions, a module can export data in public structures, called *posters*. Therefore, data, such as the robot position, can be made available for other modules and the above levels. The temporal requirements of the modules depend on the type of processing they perform. Modules running servo loop (which have to be run at precise rates and intervals without any lag) will have a higher temporal requirement than a motion planner, or a localization algorithm.

The modules are automatically generated by G$^{en}_{o}$M [Fleury 94] using a template which fields allow to define requests (parameters, failure reports, incompatibility with other requests,...), real-time constraints (period and lag if any), the type of data exported in posters and their update frequency, etc.

The functional level in Figure 4.1 details the network[2] of modules used during our tests on Dala to perform autonomous navigation tasks in an unknown environment. A navigation task implies complex processes: environment modeling, robot localization and trajectory planning. An overview of the methods developed by the LAAS to fulfill these functionalities can be found in [Lacroix 02]. Following is a brief description of the methods we used:

- **Environment modeling-** To generate elementary trajectories on rough terrains, a *digital elevation map* is computed as the robot navigates. This method uses stereo-vision data as input and represents the terrain as a set of elevations computed on a regular horizontal Cartesian grid (cf. Figure 4.3) (module LANE).
- **Robot localization-** Two algorithms are used to estimate the robot position on the basis of the on-board data: *odometry* and *visual motion estimation* (or *visual odometry*). The first one produces a position estimate at a regular high frequency but is error prone due to the wheels slippage (module RFLEX). The second one uses pixel tracking in the video images. It produces reliable position estimates, but at a variable low frequency (module STEO). A *position manager* collects these data and produces a single consistent position, thus addressing the issues of sensor geometrical distribution, asynchronism and fusion of the various position estimates (module POM).
- **Trajectory planning-** On a rough terrain, the trajectory planner has to compute a path to reach a way-point that avoids obstacles. The P3D algorithm evaluates

[2] The octagons are posters produced by the adjacent modules. An arrow indicates that a module can read and use the pointed poster.

elementary circle arcs while taking into account ground/robot collision constraints and stability constraints on the basis of the digital elevation map and the robot configuration (module P3D). The algorithm periodically reevaluates the position and chooses an arc to execute which is translated into a speed reference (executed then by RFLEX).

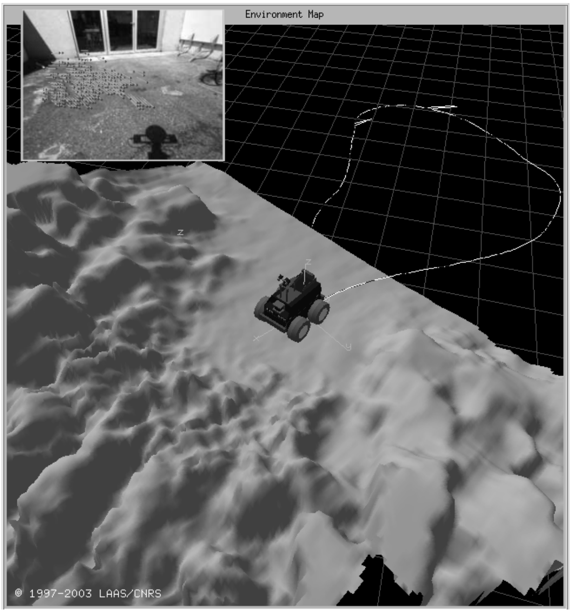


Figure 4.3: *Digital elevation map*

Table 4.1 summarizes the encapsulation of these functions in the modules.

	Module	**Description**	**Poster**
Hardware controllers	PLATINE	Pan-tilt unit controller	- PTU position
	CAMERA	Cameras controller (settings + acquisition)	- pair of stereo images tagged with the acquisition time and current robot position
	RFLEX	Odometry and servo-control	- robot position estimate
Localization	STEO	Visual odometry	- robot position estimate
	POM	Position manager	- current robot position
	SCORREL	Stereo vision (correlation procedures)	- stereo-correlated image tagged with the acquisition time and robot position
Environment modeling	LANE	Map building	- digital elevation map
Motion generation	P3D	3D local planner	- speed reference

Table 4.1: *Modules used on board Dala*

4.1.2 Requests control level

Located in between the functional and the decisional levels, the Requests and Resources Checker (R^2C) has a fault protection role [Ingrand 02]. It checks the requests sent to the modules (either from the procedural executive, but also internally from the functional level itself), as well as the resources usage. It is synchronous with the modules, in the sense that it sees all the requests sent to them, and all the reports coming back from them. It acts as a filter which allows or disallows requests to pass, according to the current state of the system (which is built online from the past requests and past reports) and according to a formal model (given by the user) of allowed and forbidden states of the functional system. When reports of the requests are being sent back to the R^2C, it passes them to the requester, after updating its internal state. The temporal requirements of this level are hard real-time.

The model of acceptable and required states is specified through a set of constraints describing contexts that block or are necessary for the execution of a particular request. These contexts can be: termination condition of the last executed instance of the request, precedence relations between requests, resource values and compatibility with requests currently in execution. The ExoGen tool compiles off-line this model into an OBDD (Ordered Binary Decision Diagram) like structure which is then used to check the specified constraints in real-time. The R^2C controls the system by rejecting or killing requests.

This component has been implemented and tested on-board Dala in parallel with our tests, but future work should encompass a complete integration of the three levels of the architecture.

4.1.3 Decisional level

The decisional level embeds the processes that require anticipation and a knowledge of the robot mission, of its global state and of the execution context. These processes are:

- The **temporal planner/executive** IxTeT-eXeC which produces plans of high-level actions achieving the mission goals, and supervises their execution and timing.
- A **procedural executive** which interacts with the operator and the functional level (possibly through the R^2C), refines the actions of the plans, supervises the execution of each action and reacts to incoming events. This executive has been implemented using the OpenPRS tool.

OpenPRS

In the context of robot experiments, the *Procedural Reasoning System* [Ingrand 96] has interesting properties:

– a high-level language allowing for the representation of goals and conditional subplans,
– the ability to deal with different activities in parallel,
– a bounded reaction time to new events is guaranteed.

OpenPRS is composed of a set of tools to represent and execute procedures:

- **A database-** It contains facts representing the system view of the world, and is constantly and automatically updated as new events appear (internal events or external events from an operator, TEXEC or the functional modules). Thus the database can contain symbolic and numerical information such as the position of the robot or the status reports of requests.
- **A library of procedures-** Each procedure describes a particular sequence of actions and tests that may be performed to achieve given goals[3] or to react to certain situations. Each procedure is self-contained: it describes in which conditions it is applicable and the goals it achieves.
- **A task graph-** It corresponds to a dynamic set of tasks currently executing. Tasks are dynamic structures which keep track of the state of execution of the intended procedures and of the state of their posted subgoals.

An interpreter runs these components: it receives new events and internal goals, selects appropriate procedures based on the new situation and places them on the task graph, chooses one task and finally executes one step of its active procedure. This can result in a primitive action (e.g. a request sent to a module), or the establishment of a new goal.

In complement, the Transgen tool allows a complete integration between OpenPRS and GenoM. It takes a list of functional modules as input and generates a PRS kernel including the basic procedures for each request and poster and all the encoding/decoding functions.

Execution control within the procedural executive

In the decisional level, the procedural executive is the only component closing the loop with the lower levels: it sends requests to the functional modules and collects the corresponding reports. This executive is also reactive to the commands sent by the operator (e.g. high-level goals) and interacts with TEXEC. TEXEC provides it with actions (belonging to a plan) to execute. In return, OpenPRS informs IXTET-EXEC about data relevant to planning: new goals, actions termination status, the system state and resources levels.

The execution control of plans is divided between the procedural and the temporal executives. On one hand, TEXEC decides when to start the high-level actions of the current

[3] In PRS, *goals* correspond to the description of a desired state along with the behavior to reach/test this state. Thus goals can be: *achieve* a condition, *test* a condition, *wait* for a condition to become true, passively *preserve* a condition or actively *maintain* a condition while doing something else. Apart from goals, the possible instructions in a procedure include explicit addition or removal of facts in the database, standard programming structures (if-then-else, while, etc.) and an extensive use of variables.

plan and when to stop them if they are preemptive. On the other hand, the procedural executive has to fulfill the following functionalities:

- **Action refinement**
  To each high-level action (fully instantiated by TEXEC) corresponds at least one procedure that expands the action into requests to the modules.[4] The library can contain several such procedures for each action, that are applicable in different contexts.

- **Action execution monitoring**
  - During the action execution, OpenPRS checks the termination report of each request.
  - The procedures corresponding to preemptive actions have to be receptive to a possible interruption message at any time. They apply in response a sub-procedure that stops the activities of the action (running modules. . .) in a clean and safe manner.
  - Once an action is finished, OpenPRS computes the report sent to TEXEC according to the reports of the requests and to the state of the system (available through posters).

- **Resources monitoring**
  OpenPRS performs a regular monitoring of the state of the resources and devices, to signal a problem to IXTET-EXEC.

- **Failure recovery**
  OpenPRS has some latitude to recover from requests failures before they get reported to IXTET-EXEC as a last resort:
  - First, all subgoals are automatically reposted until a "complete" failure is reached. By complete failure, we mean that all the applicable procedures (with all possible context bindings) have failed.
  - Second, often procedures are written in such a way that they test at run time what is the best execution path to take according to the context (requests reports, resource availability, etc.), and may recover from immediate failures.

  However, all recovery activities should be compatible with the preemption status of the action. If a recovery procedure for a preemptive action takes some time and can not be interrupted, it should result from an explicit planning process and correspond to a non preemptive action in the plan.

- **Standby mode**
  Finally, OpenPRS should have a standby procedure to rely on while plan execution is aborted and replanning is in progress. In our case, the robot just stays still.

[4] A TakePicture action, for instance, corresponds to the sequence of requests `OneShot` (take a raw image) and `Save` (compress and store the image in a specific location) sent to the Camera module.

4.2 Models

The exploration mission scenario requires the robot to accomplish three types of goals: take pictures of specific targets, communication with a ground station during visibility windows and a return near the lander at the end of the mission. These goals can be achieved through five main high-level actions (cf. section 1.5):

- move between two absolute locations,
- take an image,
- move the pan&tilt unit from a position to another one,
- download images to the lander,
- communicate with a ground station during its visibility window.

The first three actions can be physically performed by Dala. The execution of the last two actions however can only be simulated on-board the robot. This is done through specific PRS procedures which simulate the visibility windows or the gradual download of images. We briefly present the models developed for the experiment.

4.2.1 IxTeT-eXeC model

This model is given in Appendix 2. It contains 9 state variables (including the x and y position of the robot, etc.), 2 unary resources (e.g. the camera bench) and 1 reservoir resource (memory storage). 6 rigid attributes represent diverse characteristics of the domain: the speed of the robot (between 0.03 and 0.1 $m.s^{-1}$), the capacity of memory storage[5], the download rate, the image compression rate, etc.

The model contains 9 actions in all (cf. their description in section 1.5):

- move($?x_1, ?y_1, ?x_2, ?y_2$), movex($?x_1, ?y_1, ?x_2, ?y_2$) and movey($?x_1, ?y_1, ?x_2, ?y_2$) are *latePreemptive* actions. They correspond to the navigation between the absolute locations ($?x_1, ?y_1$) and ($?x_2, ?y_2$).[6] The duration of the action is estimated according to the speed of the robot and to the Manhattan distance between the two locations.
- move_pan_tilt_unit(?initpos,?finalpos) is a *latePreemptive* action.
- take_picture(?obj,?x,?y) is a *nonPreemptive* action. The image compression rate has been defined according to a sample of 40 images taken during tests.
- download_images() is a *latePreemptive* action. It is intended to completely free the memory, thus its theoretical duration is defined according to the size of the directory and to the download rate.

[5] We simulate a limited storage capacity (66 kB): the images are stored in a specific directory on Dala which size should never exceed the chosen capacity.

[6] The 3 actions correspond to an equivalent execution procedure. The actions movex and movey along the x and y axis are only given to improve the performances of the planner.

- communicate() is a *latePreemptive* action. It is constrained to occur during a contingent temporal window and the robot should not move.

- init_ptu_driver() and init_mvt_generation() are *nonPreemptive* actions. They correspond to recovery actions used when move_pan_tilt_unit and move fail with specific failure reports. The first one implies a reinitialization of the Platine module, the second one implies the reinitialization of the Lane and P3D modules. These processes should not be interrupted.

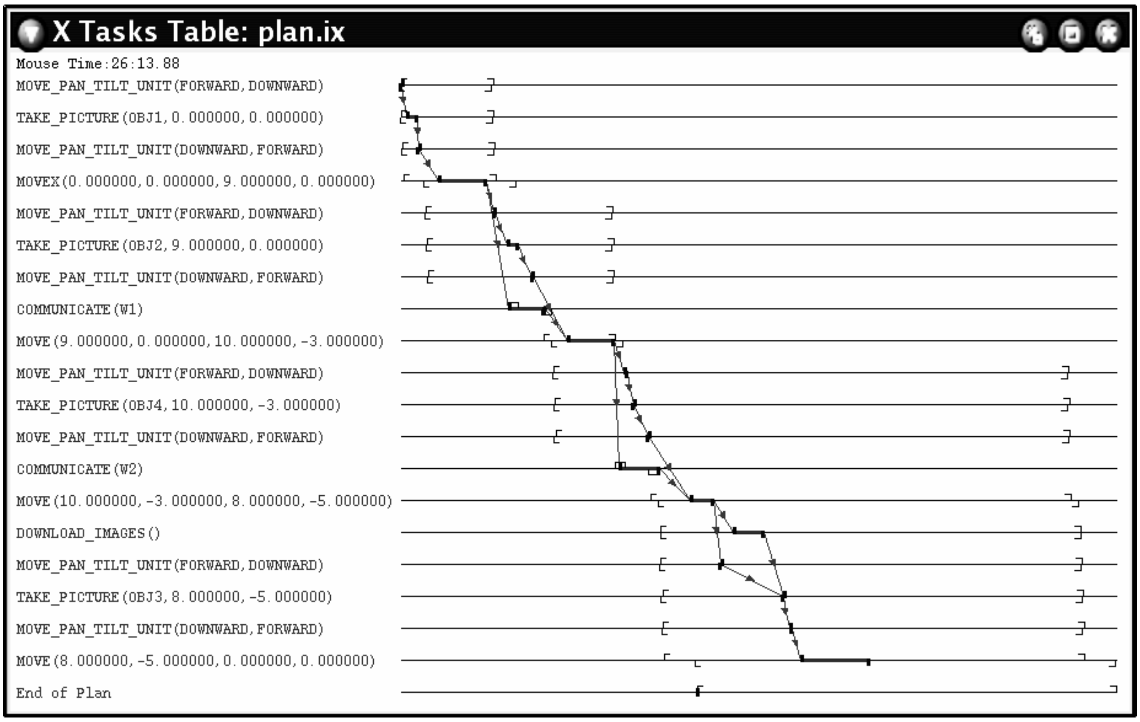
Figure 4.4 shows the plan generated for the following initial scenario:

– **Initial situation:** the robot is in (0,0); the storage directory is empty; the cameras orientation is “forward”.

– **Goals:** 4 images in (0,0), (9,0), (10,-3), and (8,-5); 2 communications during the visibility windows w_1 and w_2; be back in (0,0) at the end of the mission.

– **Temporal constraints:** the window w_1 starts at $t = 300s$ and lasts $120s$; the window w_2 starts at $t = 600s$ and lasts $120s$; the planning horizon is $2000s$.

This initial plan contains 19 actions and 57 timepoints.



X Tasks Table: plan.ix

```
Mouse Time:26:13.88
MOVE_PAN_TILT_UNIT(FORWARD,DOWNWARD)
TAKE_PICTURE(OBJ1,0.000000,0.000000)
MOVE_PAN_TILT_UNIT(DOWNWARD,FORWARD)
MOVEX(0.000000,0.000000,9.000000,0.000000)
MOVE_PAN_TILT_UNIT(FORWARD,DOWNWARD)
TAKE_PICTURE(OBJ2,9.000000,0.000000)
MOVE_PAN_TILT_UNIT(DOWNWARD,FORWARD)
COMMUNICATE(W1)
MOVE(9.000000,0.000000,10.000000,-3.000000)
MOVE_PAN_TILT_UNIT(FORWARD,DOWNWARD)
TAKE_PICTURE(OBJ4,10.000000,-3.000000)
MOVE_PAN_TILT_UNIT(DOWNWARD,FORWARD)
COMMUNICATE(W2)
MOVE(10.000000,-3.000000,8.000000,-5.000000)
DOWNLOAD_IMAGES()
MOVE_PAN_TILT_UNIT(FORWARD,DOWNWARD)
TAKE_PICTURE(OBJ3,8.000000,-5.000000)
MOVE_PAN_TILT_UNIT(DOWNWARD,FORWARD)
MOVE(8.000000,-5.000000,0.000000,0.000000)
End of Plan
```

Figure 4.4: *Initial plan*

4.2.2 OpenPRS model

The procedures of the model can be sorted in 3 main sets:

1. the procedures in connection with the navigation task,
2. the procedures linked to the joint plan execution with TEXEC,
3. the procedures used to simulate the download and communication actions.

Navigation

To perform a move action, we use the library developed by G. Infantes. In brief, there are multiple ways to perform a navigation task (depending on the methods used for localization, map building, motion generation). Each such way, also called *modality*[7], can be modeled as a *Hierarchical Task Network* whose primitives are sensory-motor functions. An automatic generation ensures the consistency of these modalities by taking into account the data flow between functions (e.g. video images). Further details can be found in [Infantes 03]. Few modalities are available for outdoor navigation on rough terrains and we mainly performed our tests with the modality based on stereo odometry and the P3D trajectory planner.

Mission

This set contains:

- Procedures to communicate with the temporal executive such as the following ones, fired by the LAUNCH and END commands sent by TEXEC.

```
(defop |start-task|
  :invocation (LAUNCH $REQUESTER $TASK_NAME $TASK_ID $PARAMS)
  :body
  (
   (!($TASK_NAME $TASK_ID $PARAMS $RETURN))
   (!(SEND-MESSAGE $REQUESTER (BILAN $TASK_ID $RETURN)))
  )
)

(defop |end-task|
  :invocation (END $TASK_ID)
  :body
  (
   (=> (INTERRUPT $TASK_ID)))
  )
)
```

- One "action refinement" procedure for each high-level action. Following is an example of "refinement" procedure for the preemptive move_pan_tilt_unit action. Such

[7] The different modalities are more or less appropriate depending on the environment (cluttered, dynamic, narrow, etc.). The relationship between the execution context and the appropriate modality can be learned from experiment as a *Markov Decision Process* which provides a general policy for the navigation task. This work is detailed in [Morisset 02] as well as the thorough experiments made on an indoor robot.

a procedure achieves at least two roles: the expansion in requests to the modules (PLATINE-CMDPOSCOORD-REPORT, PLATINE-STOP-REPORT) and the computation of the report sent to TeXeC ($RETURN).

```
(defop |task-move-pan-tilt-unit|
  :invocation (!(MOVE_PAN_TILT_UNIT $TASK_ID (PARAMETERS $POSI $POSF) $RETURN))
  :effects ((~> (INTERRUPT $TASK_ID)))
  :body
  (
   (? (PAN_ANGLE $POSF $P_ANGLE))
   (? (TILT_ANGLE $POSF $T_ANGLE))
   (// ((^ (INTERRUPT $TASK_ID))
        (! (PLATINE-STOP-REPORT  $STOP_PTU_REPORT))
       )
       ((! (PLATINE-CMDPOSCOORD-REPORT
            (PLATINE_REF_COORD_POS_STR
            (UNITY PLATINE_DEG)
            (CMDTYPE PLATINE_ABSOLUTE)
            (PAN $P_ANGLE)
            (TILT $T_ANGLE)
            (SURVENDFLAG PLATINE_TRUE)
            (UNUSED 0))
            $MOVE_PTU_REPORT))
        (IF (? (EQUAL $MOVE_PTU_REPORT "OK"))
            (! (= $RETURN (NOMINAL)))
            (=> (INTERRUPT $TASK_ID))
         ELSE
            (IF (? (INTERRUPT $TASK_ID))
                (! (= $ENDSTATUS INTERRUPTED))
             ELSE
                (! (= $ENDSTATUS FAILED))
                (=> (INTERRUPT $TASK_ID)))
            (IF (? (EQUAL $MOVE_PTU_REPORT "DRIVER_NOT_INITIALIZED"))
                (!(= $ATT_INIT (PTU_DRIVER_INITIALIZED F)))
             ELSE
                (!(= $ATT_INIT (PTU_DRIVER_INITIALIZED T)))
            )
            (! (GET_PTU_POS $PTU_POS))
            (!(= $ATT_POS (PAN_TILT_UNIT_POSITION $PTU_POS)))
            (! (= $RETURN ($ENDSTATUS (STATEBILAN $ATT_POS $ATT_INIT))))
         )
       )
     )
    )
)
```

- Miscellaneous procedures to monitor resources, check the robot or pan&tilt unit position, etc.

Simulation

- **Limited memory storage-** Once compressed, an image is stored in a specific directory. However the image is deleted and the take_picture action fails if there is not enough space in this directory with respect to the specified capacity limit. A download action consists in transferring the images from this directory to another one. Before each file transfer, the procedure waits for a certain duration computed according to the file size and a specified download rate. The storage level returned at the end of the take_picture and download actions to TEXEC is the current size of the storage directory and thus reflects the uncertainty on the actual size of the images.
- **Visibility windows-** A specific procedure simulates the start and exit of the visibility windows. The communicate procedure checks that the window remains active during the action execution.

In the following section, we present examples of runs where no specific failure has been injected in the simulated part of the mission.

4.3 Runs

Dala has been given an initial plan (Figure 4.4) to execute autonomously. Each resulting run was different (in the number of actions, mission duration, etc.). We illustrate the performances of IXTET-EXEC with three of these runs. For each run: *timestep* $= 2s$ and $\mu = 60\%$. Figures 4.6, 4.8 and 4.10 represent the execution traces. They give for each cycle of TEXEC: the messages[8] exchanged with OpenPRS and the actions that are postponed when a plan repair is in progress. In complement, Figures 4.7, 4.9 and 4.11 indicate for each cycle: its starting time, its total duration and the duration of each phase of the cycle.

Run 1

During this run, the robot has achieved the trajectory shown in Figure 4.5. All goals have been accomplished in a quite straightforward manner. There has been three execution events requiring a plan repair.

1. Taking too much time, the move(9,0,10,-3) action has been interrupted in cycle 12. The report is received too late, thus leading to a time-out situation where some temporal flexibility is left. The plan repair process (40 planning steps distributed on 3 cycles) inserts a new move action to complete the navigation, but after the communication with the ground station, thus respecting its deadline.

[8] In bold: the launch or interruption of an action. In italic: the report sent back at the end of the action.

2. The take_picture($O_4, 10, -3$) action lasted less than expected in the model. The plan repair in this case mainly rebuilt the causal links removed during the message integration (done in 29 planning steps and 3 cycles).

3. The return to the position (0,0) takes longer than expected in the model, thus move(8,-5,0,0) is interrupted in cycle 29. The plan repair process inserts a move action to complete the navigation (in 12 planning steps and 2 cycles).

On a run of 1370s, IxTeT-eXeC woke up 33 times and used 640s of CPU time. The other main processes: STEO, CAMERA, SCORREL, OpenPRS, LANE and P3D, have respectively used 271s, 144s, 119s, 75s, 52s and 20s of CPU time.

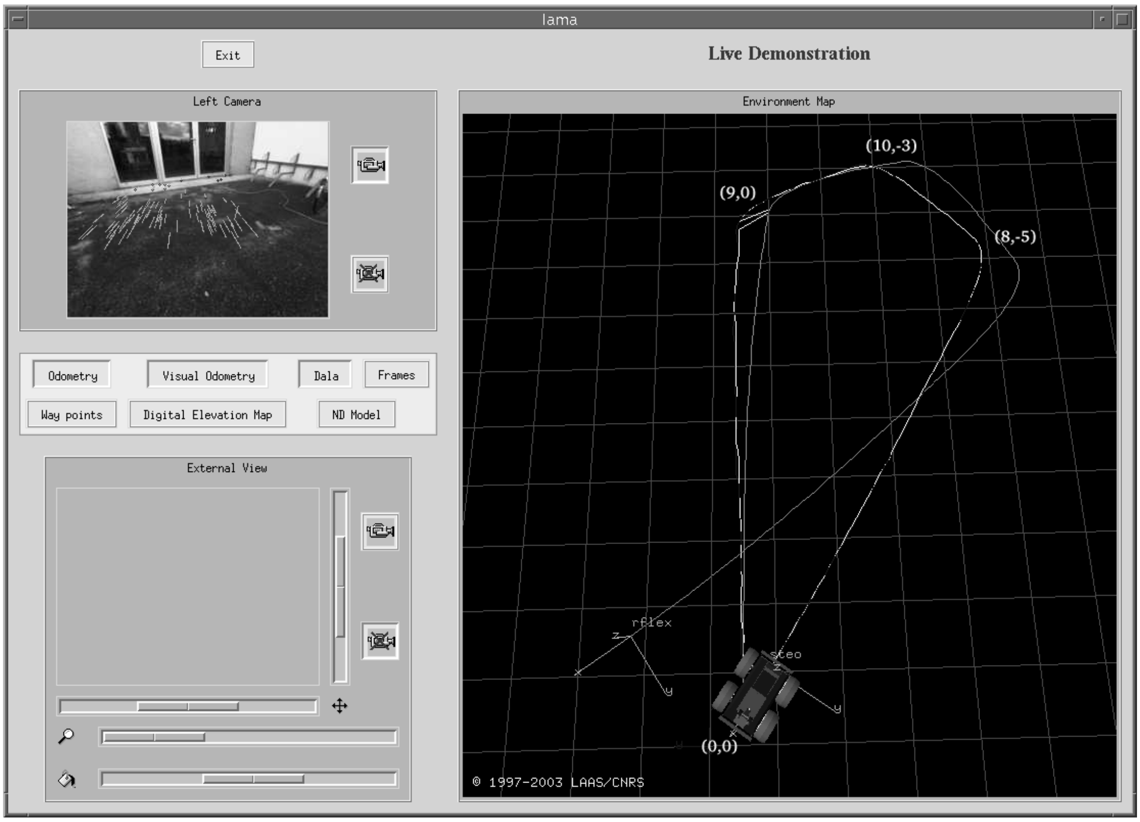


Figure 4.5: *Path covered by Dala during Run 1. The trajectories labeled "rflex" and "steo" represent the position estimated respectively by odometry and visual odometry along the path. The visual odometry is indispensable even if it considerably slows down the robot.*

Run 2

Some unknown obstacles have been added (approximately in (3,-1) and (6,1)). All goals have been achieved. Two execution events required a plan repair:

1. Due to the obstacles, the movex(0,0,9,0) action takes much more time than expected. It is interrupted in cycle 5. Not much flexibility is left before the visibility window w_1, still the plan repair finds a solution (in 50 planning steps and 4 cycles) before the starting timepoint of the communicate action times out. A move action is inserted after the communication.
2. Here again, a take_picture action takes less time than expected. The plan repair process (39 planning steps) is distributed on 4 cycles (14 to 17). Note that a move_pan_tilt_unit action is launched as soon as it is supported in the plan (cycle 16), and before a solution plan is found.

On a run of 1285s, IxTeT-eXeC woke up 32 times and used 541s of CPU time. The other main processes: STEO, CAMERA, SCORREL, OpenPRS, LANE and P3D, have respectively used 270s, 145s, 123s, 80s, 55s and 18s of CPU time.

Run 3

This time, the obstacles have been placed approximately in (3,1) and (6,-2). The robot did not achieve all initial goals. The following events occurred (a detailed trace of this run is given in Appendix 2):

1. Similarly to the previous run, the first move action takes too long and is interrupted in cycle 5. The plan is repaired with the insertion of a move action after the communication (in 51 planning steps).
2. However, this second move action takes also too much time and is interrupted (cycle 12). This time, no temporal flexibility is left in the plan (due to the temporal constraints of the visibility window w_2) and IxTeT-eXeC completely replans with the abandon of the goal requiring a picture in (10,-3). The new plan contains a communication in w_2, a travel to (8,-5) to take a picture, a return to (9,0) to take a picture and a final return to (0,0).
3. The execution of the move(8,-5,9,0) action takes a lot of time due to the obstacles. It is interrupted and a new navigation action is replanned several times (cycles 25, 28, 31).
4. Finally, the robot is stopped while it is going back to (0,0) because we had specified a too short mission horizon :-(.

On a run of 2000s, IxTeT-eXeC woke up 39 times and used 881s of CPU time. The other main processes: STEO, CAMERA, SCORREL, OpenPRS, LANE and P3D, have respectively used 448s, 227s, 193s, 80s, 24s and 23s of CPU time.

When no plan repair is required, the execution control rate is much better than 0.5Hz: the average duration cycle (with no plan repair) was 0.169s during Run 1, 0.19s during

Run 2 and 0.095s during Run 3. A message integration takes more time when the storage level is updated. For example, during Run 1, the resource update takes respectively 0.196s, 0.147s, 0.173s, 0.255s and 0.238s in the cycles 3, 7, 20, 26 and 27.

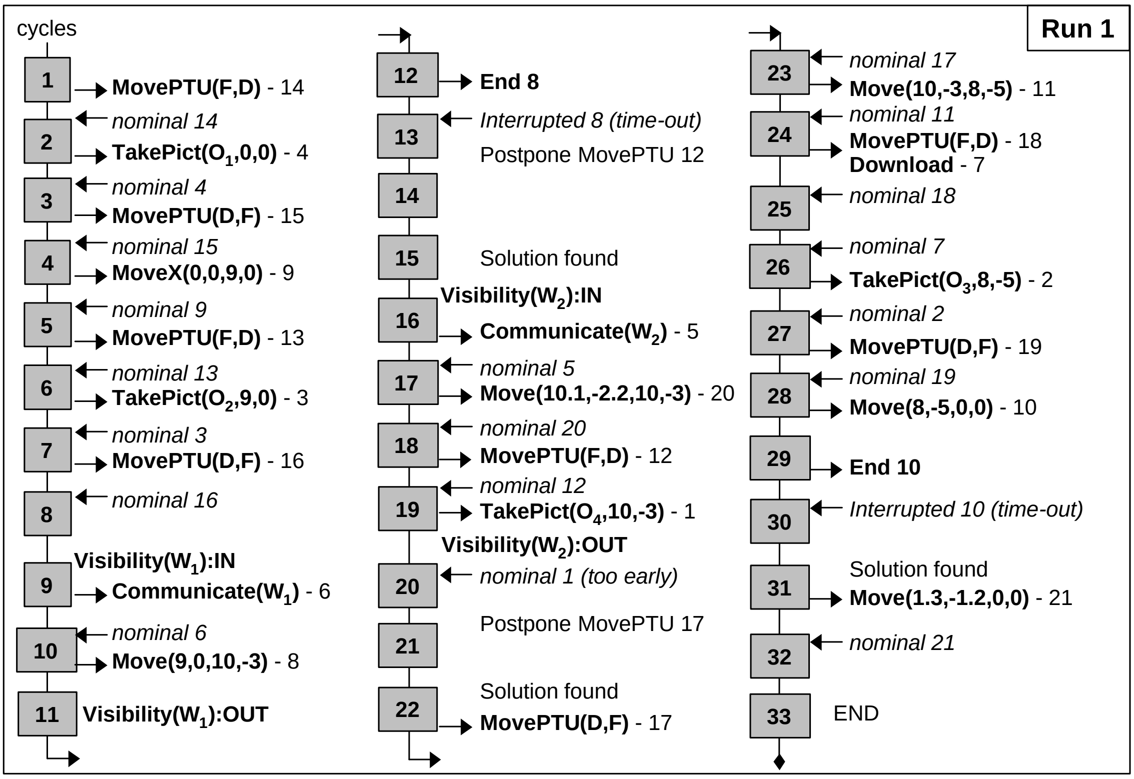


Figure 4.6: *Run 1 - Execution trace*

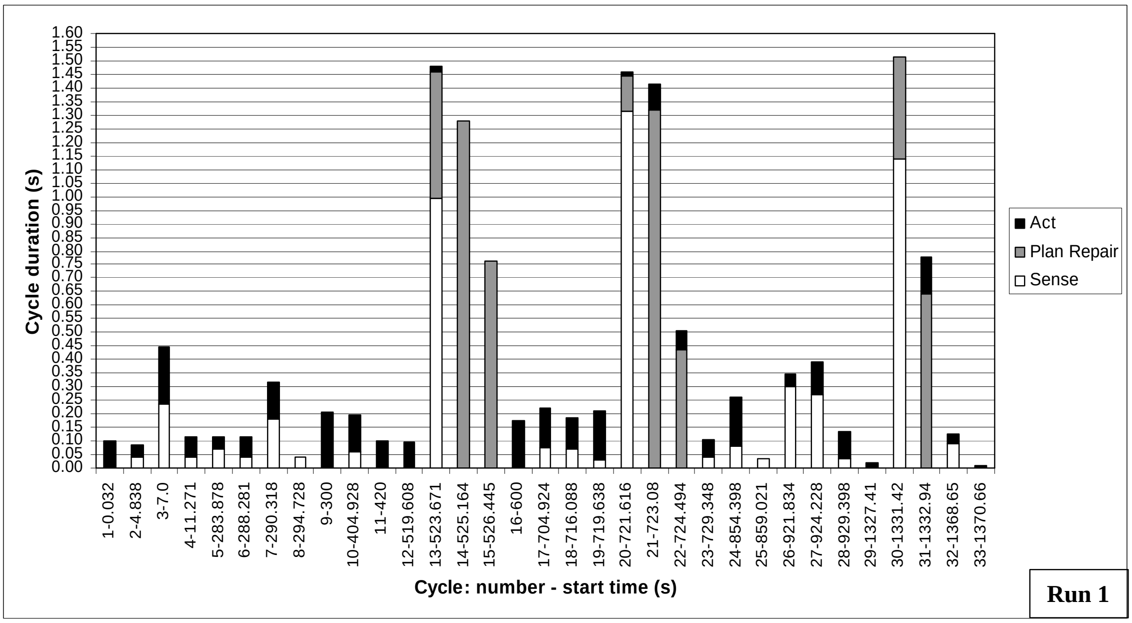


Figure 4.7: *Run 1 - Cycle start time and duration*

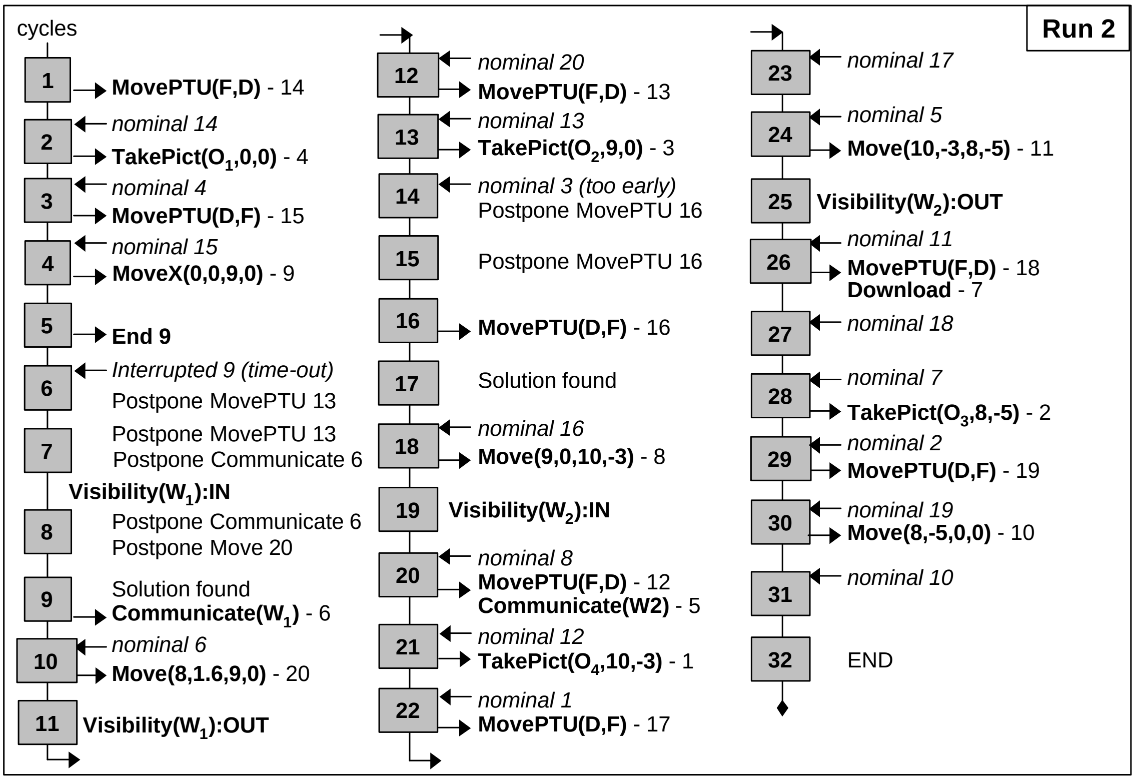


Figure 4.8: *Run 2 - Execution trace*

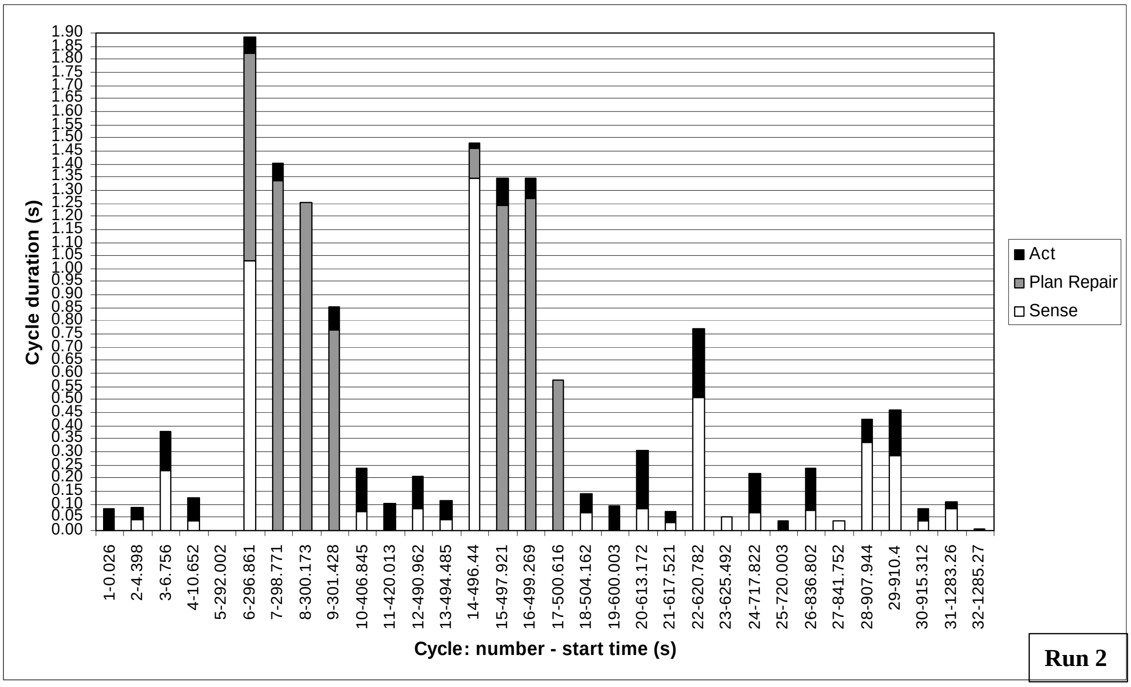


Figure 4.9: *Run 2 - Cycle start time and duration*

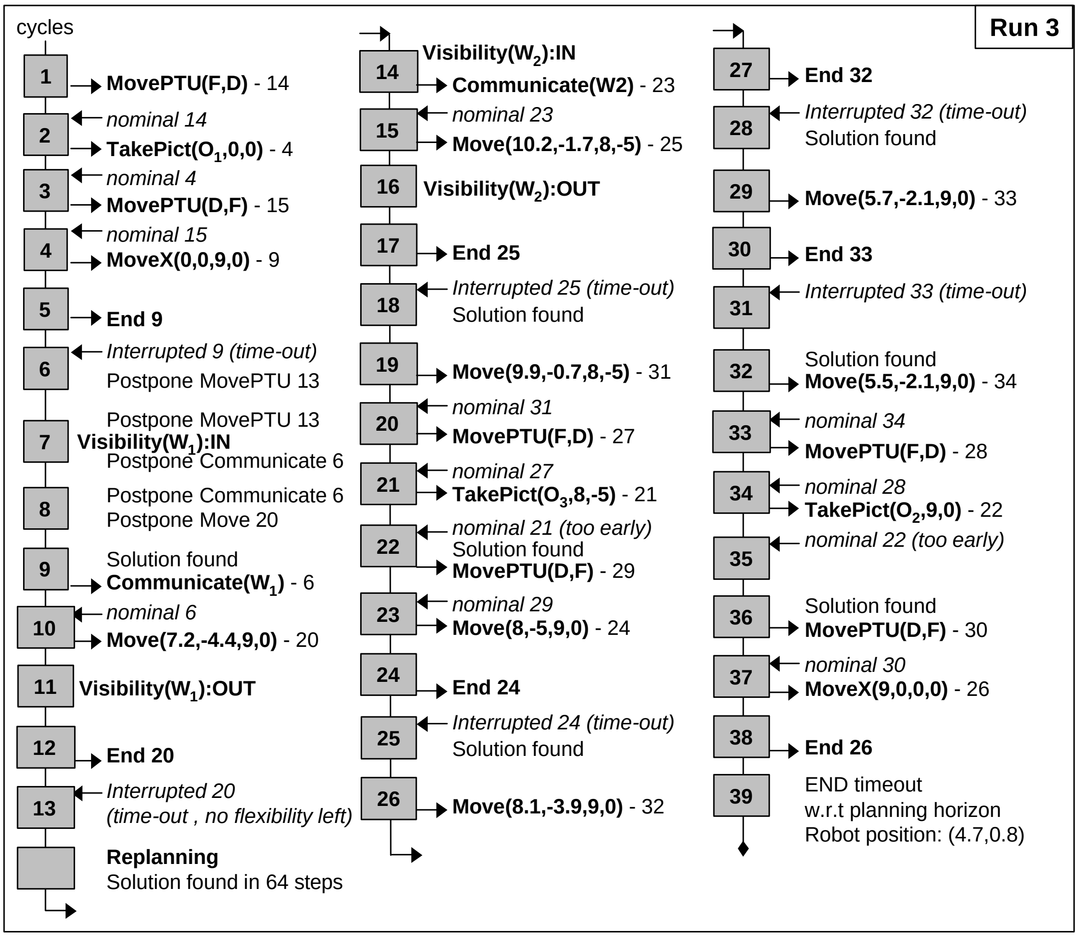


Figure 4.10: *Run 3 - Execution trace*

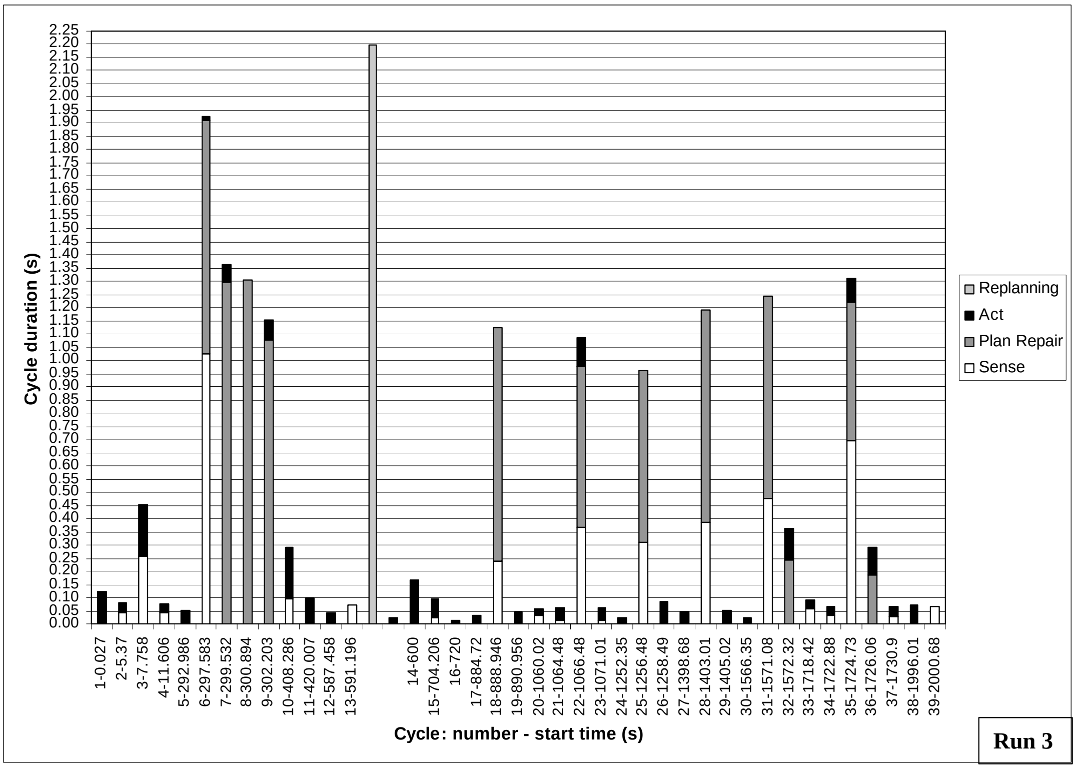


Figure 4.11: *Run 3 - Cycle start time and duration*

Figures 4.12 and 4.13 show the evolution of the memory storage level during Run 1 and Run 2. Measures are done:

– at the beginning of the mission (cycle 1),

– after each picture in (0,0), (9,0), (10,-3) and (8,-5) (cycles 3, 7, 20, 27 in Run 1 / cycles 3, 14, 22, 29 in Run 2),

– after the download action (cycle 26 in Run 1 / 28 in Run 2).

The points linked by lines correspond to the minimal and maximal levels authorized in the initial plan (cf. Figure 2.30, page 93). The circles and crosses show how these minimal and maximal "theoretical" levels evolve when updated during execution. Finally, the squares represent the real level at the end of each action. Thus, at each action end, the real level is compared with the corresponding theoretical bounds, which are then updated accordingly.

The extension to a flexible model of resources proposed in section 2.4 allows to handle the uncertainty on the actual usage of the resource at execution time. This usage varies with each run, but a plan repair becomes necessary only if the level is outside of the theoretical bounds for which the plan is guaranteed valid and if a conflict is detected (an over-consuming maximal clique).

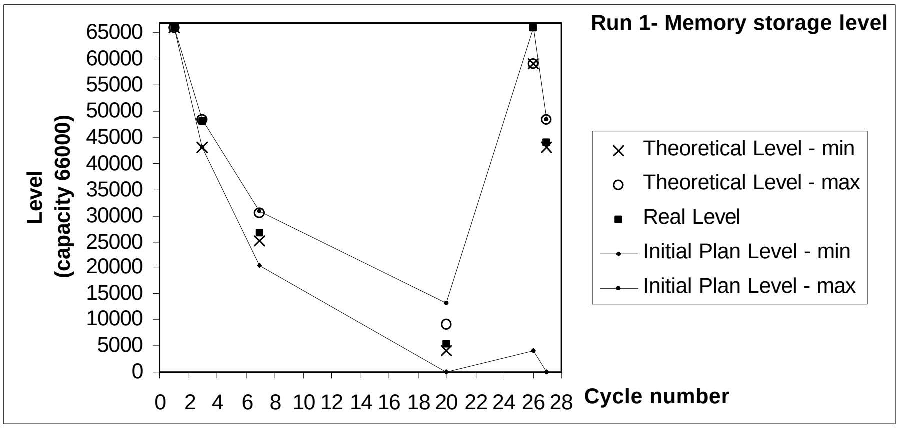


Figure 4.12: *Run 1 - Evolution of the storage level*

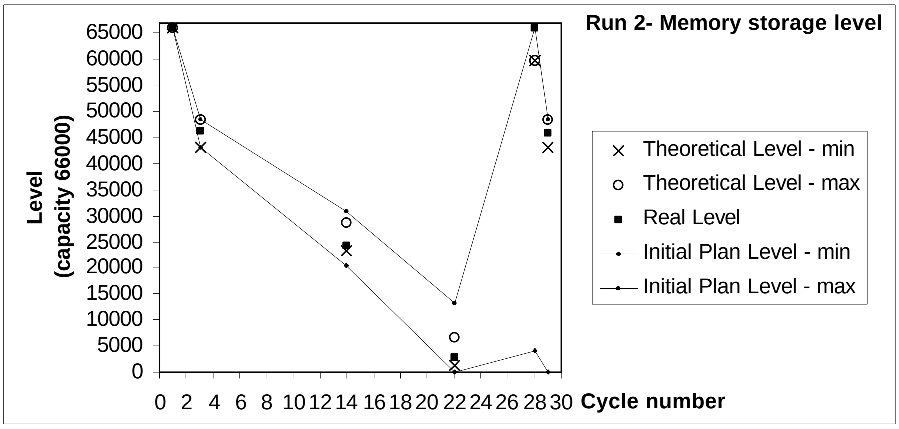


Figure 4.13: *Run 2 - Evolution of the storage level*

Conclusion

The flexibility of durations and resource usages provides an initial plan robust to execution contingencies. Furthermore, the plan repair process allows a quite reactive response to numerous failures. Consider for instance the interruption of move(0,0,9,0) in Run 2. The plan repair takes 4 cycles, i.e. much less than the time needed for a complete replanning at that state of the mission execution (with 6 goals left including 3 images and 2 communications)[9]. In the above runs, the communication windows are respected despite the delays of the robot.

We have encountered several types of failures during our tests, some of which could be repaired, some other could not. For example, the Platine module has returned a "DRIVER_NOT_INITIALIZED" report a few times which could be successfully repaired by the reinitialization of the module. More often, the P3D module has returned a report notifying that the robot was blocked even if no obstacle were present on its way. In such cases, the reinitialization of the digital elevation map and of the P3D module sometimes solved the problem. It also happened that the brakes were put on by the controller for no apparent reason, thus stopping the robot. This problem was not reported by any modules and could not be handled by the above levels.

Our experiments rely on other algorithms which have not necessarily been implemented with the aim of being used by a supervisor in a completely autonomous mode. Diagnostic and recovery activities need to be enforced at any level of the architecture. The IxTeT-eXeC system is a just step towards a more robust autonomous behavior w.r.t. the mission temporal and resource constraints.

[9] The initial plan (for 7 goals including 4 images and 2 communications) is found in 79.6s, 126 developed nodes and 8 backtrack nodes.

Chapitre 5

Ce chapitre présente des résultats supplémentaires obtenus par simulation du comportement du système contrôlé grâce à des procédures spécifiques sous OpenPRS.

Domaine

Les exemples présentés se basent sur un nouveau domaine : un robot chargé d'explorer une zone inconnue et de ramener les objets découverts en chemin.

Entrelacement de la réparation et de l'exécution du plan vs Replanification

Nous comparons les deux stratégies lorsque le robot découvre de nouveaux buts ou doit réagir à deux échecs consécutifs : (1) entrelacement de la réparation et de l'exécution du plan, (2) arrêt de l'exécution et replanification complète. Nous analysons plus particulièrement le temps global passé dans l'exécution des actions du plan, dans les phases "Perception" et "Action", et en planification.

Ressources

Le modèle prend maintenant en compte l'énergie consommée par le robot. Nous montrons comment le système réagit à des variations plus ou moins importantes du niveau par rapport aux prédictions, et à une chûte de la capacité.

Chapter 5

Illustrative example

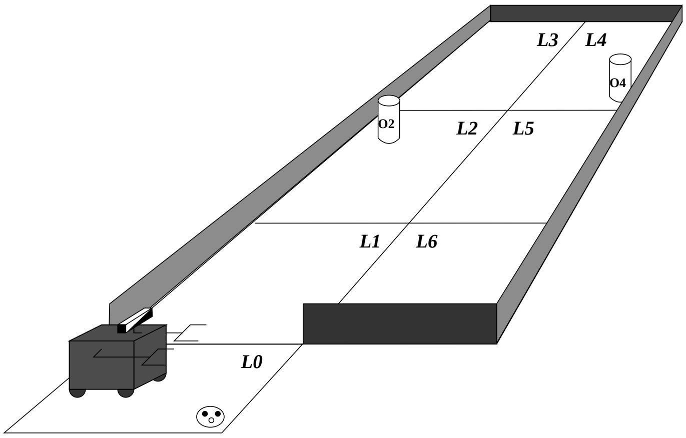
IxTeT-eXeC provides two plan adaptation mechanisms (plan repair / replanning) in response to the following situations:

1. temporal failures,
2. failures of actions,
3. new goals,
4. detection of future resource contention.

The runs on Dala presented in the previous chapter show how the system can react to temporal failures when deadlines have to be respected. In this chapter, we illustrate the reactions to the other situations. We use a new domain example: a robot has to explore an unknown area and bring back objects that it discovers on its way. IxTeT-eXeC is interfaced with OpenPRS and the behavior of the robot is simulated by specific procedures.

The first section presents the domain. The second section aims to compare the plan repair and replanning strategies in this context. In the last section, we consider how the energy consumption can be handled by the system.

5.1 Domain



A robot, equipped with two arms (LH and RH) and initially located in L_0, is required to explore an area divided into 6 locations ($L_1 \ldots L_6$). The robot takes images in each location, analyses them and brings back (to L_0) any object present in the location and revealed by the image processing.

The robot capabilities are described as a set of 5 actions:

- move($?l_i$,$?l_f$) - move from location $?l_i$ to location $?l_f$,
- scan($?l$) - take images in location $?l$ and process them,
- take($?o$,$?l$,$?h$) - take the object $?o$ in location $?l$ with the arm $?h$,
- put($?o$,$?l$,$?h$) - put the object $?o$ on the ground in location $?l$ with the arm $?h$,
- carry($?o$,$?l_i$,$?l_f$,$?h$) - carry the object $?o$ from location $?l_i$ to location $?l_f$ with the arm $?h$ (this action can be executed in parallel with the other actions: it checks that the object is kept in the arm during the travel).

The move action is late preemptive. Scan, take and put are non preemptive actions. Carry is an early preemptive action: it will be terminated as soon as the robot arrives in the final location with the object. The initial goals of the mission are: “scan each location” and “be back in L_0 at the end of the mission”. Thus the initial plan is a sequence of alternated move and scan actions. New goals appear during the mission execution when objects are discovered. Such a goal requires an object to be in L_0 at the end of the mission. For the following tests, we suppose that there is at most one object O_i in L_i ($i = 1..6$). No specific temporal constraints are required.

The tests have been run on a Pentium IV. The simulated durations of the actions are: 3s for scan and 8s for put and take. The duration of the move action varies with the distance (10s for (L_0, L_1), 20s for (L_2, L_4), 30s for (L_1, L_4), etc.). During the execution of a move action, the simulated position of the robot is set to an intermediate location (L_1 for (L_0, L_2), L_1 for (L_1, L_2), etc.).

5.2 Plan repair interleaved with plan execution vs replanning

Using this domain example, we compare the two strategies of plan adaptation:

- interleave plan repair and plan execution (***PR&E***),
- abort the plan execution and replan from scratch (**Replan**).

For each planning strategy, Figure 5.1 presents the total duration of the mission execution when the following sets of objects are present in the area: $\emptyset$ (case (0)), $\{O_1\}$ (case (1)), $\{O_1,O_2\}$ (case (2)), …, $\{O_1,O_2,O_3,O_4,O_5,O_6\}$ (case (6)). Case (0) corresponds to the nominal execution of the initial plan. In all other cases, the *PR&E* strategy leads to a shortest mission. Though, it is interesting to decompose this global duration and detail the share of each subprocess. This duration can be decomposed in:

1. The time spent in the execution of the actions by the simulator while IxTeT-eXeC is idle (labeled "Execution" in the figure).
2. The time spent in the "Sense" and "Act" phases of the execution cycles.
3. The total time spent in planning. In the *PR&E* case, it corresponds to the time spent in the "plan repair" phases of the execution cycles. In the figure, this total time is split in two parts, labeled "Execution & plan repair" and "Planning", depending whether an action is concurrently executed or not.[1]

Let us detail for each subprocess where the differences between the strategies lie in.

- The quality of the plan (in terms of makespan) varies with the planning strategy. In cases (1) and (2), the *PR&E* strategy inserts an immediate grasp of each object which are carried while the robot goes on with its scanning mission. The objects are put on the ground at the end of the mission. The *Replan* strategy produces almost the same plan, but the ordering of the visited locations is modified, thus increasing the total distance. In the other cases, the *PR&E* strategy brings back the additional objects one by one. The *Replan* strategy produces plans significantly different but not necessarily better (no optimization can be done by the IxTeT planner). The makespans of these final plans remain comparable.

- More time is spent in the "Sense" phase of the cycles with the *Replan* strategy. This is due to the execution abortion before replanning. Actions are interrupted and the system integrates non nominal reports. The most expensive process in this case is the possible relaxation of the ending timepoint of the action (which implies a recomputation of the temporal network).

- More time is spent in the "Act" phase of the cycles with the *PR&E* strategy. The main reason is that the number of timepoints in the temporal network can only grow using this strategy (past timepoints are not removed and new actions are inserted by plan repair). The propagation in the STN at each timepoint execution takes progressively more time. Whereas replanning from scratch leads to a network containing mainly the horizon and future actions/goals timepoints.

- The total time spent in planning is lower with the *PR&E* strategy than with the *Replan* strategy in cases (1), (2) and (3); similar in case (4); and greater in cases (5) and (6). We detail case (6) in Table 5.1.

[1] Thus, the planning duration can be read on the figure by summing the "Execution & plan repair" and "Planning" durations. Likewise, the makespan of the plan corresponds to the sum of the "Execution & plan repair" and "Execution" durations.

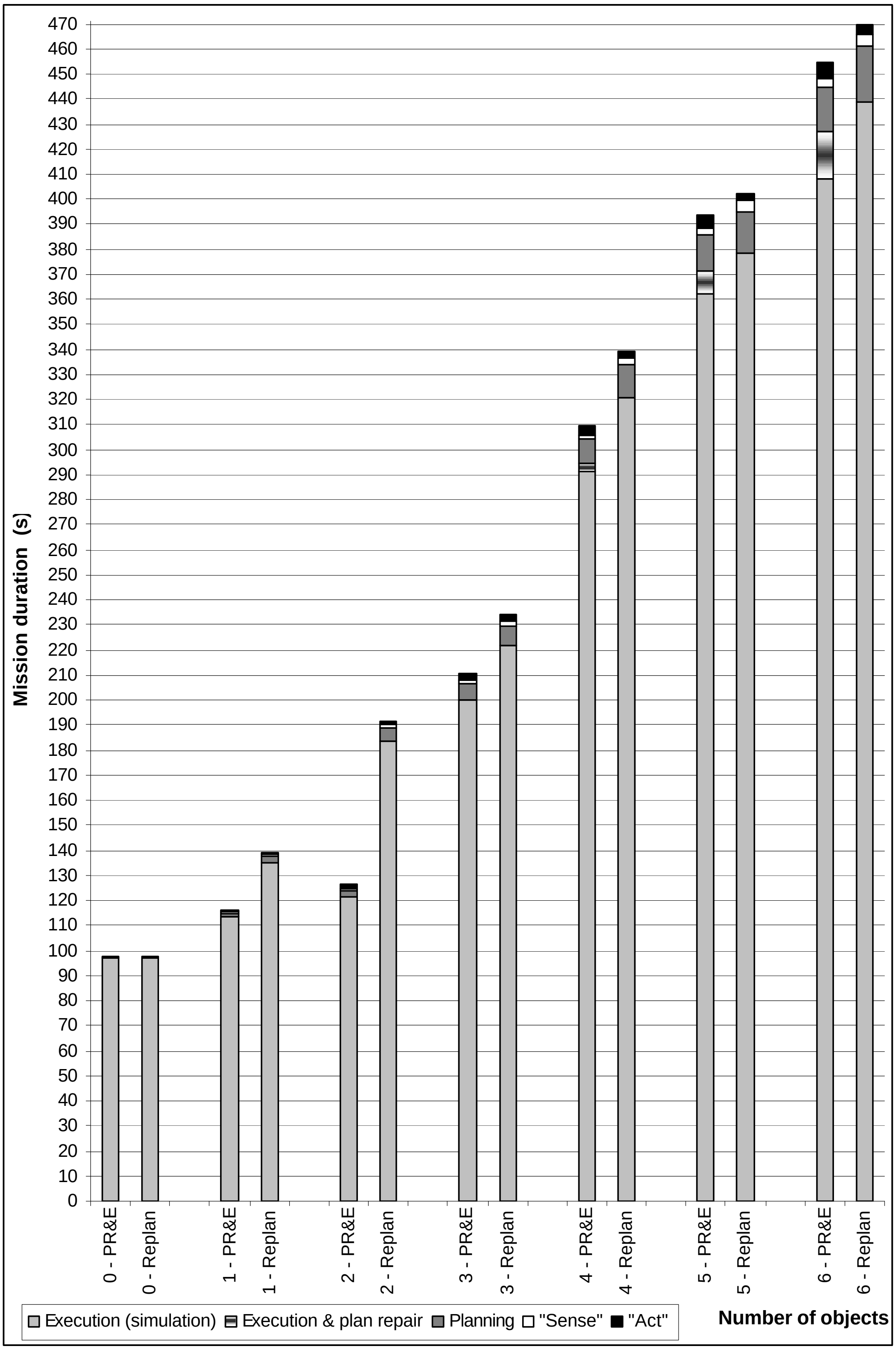


Figure 5.1: *Mission duration when 0 to 6 objects are present in the area, for each planning strategy*

This table details the planning process after each goal insertion for both strategies: the planning duration, the corresponding number of planning steps, the number of timepoints in the solution plan, and, for the *PR&E* strategy, the number of cycles and search trees ($timestep = 1s$, $\mu = 50\%$). The second search tree is due to the execution of goal timepoints. The third search tree is required because a move action is launched as soon as it is supported in the plan (leading to concurrent plan execution and repair).

At the beginning of the plan (insertion of the first and second goals), plan repair takes less time than replanning: there are less decisions to take and the average duration of a planning step is equivalent with both strategies.

After awhile however, plan repair is much more expensive than replanning.

- Plan repair implies two additional processes: checking the executability status of the plan at each planning step; and recovering the last executable plan at the end of the plan repair phase of each cycle. But these processes do not have a significant influence on the global planning duration.
- During the execution, past actions are partially "forgotten": their propositions will not be considered during plan repair. However their timepoints are not removed from the temporal network. And the average duration of a planning step increases drastically with the number of timepoints.

	PR&E				**Replan**		
New Goal	(c, st) c: Nb of cycles st: Nb of search trees	Nb of planning steps	Plan repair duration (s)	Nb of timepoints in solution plan	Nb of planning steps	Planning duration (s)	Nb of timepoints in solution plan
1^{st}	(4,2)	52	1.56	52	74	2.70	50
2^{nd}	(4,2)	40	1.57	63	76	3.27	60
3^{rd}	(8,2)	52	4.11	78	105	5.79	53
4^{th}	(11,3)	63	6.14	93	90	5.02	56
5^{th}	(17,3)	73	9.48	108	75	3.13	50
6^{th}	(23,3)	82	13.71	123	65	2.26	42

Table 5.1: *Plan repair vs replanning*

Still, plan repair interleaved with plan execution is competitive. Actions can be launched before the end of the planning process, and events are taken into account during the planning process. Consider for instance the following sequence of events: the area contains two objects (O_1 and O_2); the robot carries both objects and explores the remaining locations; 5s after the start of the last move action towards L_0, the robot lets O_1 fall on the ground; 1.7s later, it lets the second object fall on the ground.

Figure 5.2 shows the behavior of the system with both planning strategies. In the *Replan* case, the system is notified of the first event, aborts the execution of the current actions and replans. The second event happens during the replanning process. The system is notified only when it tries to execute the solution plan, and has to wait for the termination of the non preemptive action take before starting a new replanning process. In the *PR&E* case, the system is notified of the first event and starts plan repair. It interrupts the move(L_6,L_0) action which is threatened by the event. It integrates the second event as it happens, as well as the move termination report. It launches a take action as soon as it is supported in the plan. A solution plan is finally found in the 6^{th} cycle ($timestep = 1.5s$, $\mu = 50\%$). The global reactivity to both events is much better with the *PR&E* strategy even if the planning duration is longer.

It is difficult to draw general conclusions on the most appropriate planning strategy. It varies with the type of domain and the execution situations. In the given example, replanning is efficient since a complete plan can be found quickly and there is little parallelism in the plan. In IxTeT-eXeC, both strategies can be used. Further work needs to be done to define when the system has to switch from one strategy to the other one.

Plan repair is useful when the plan contains actions that can be executed independently. It is also useful to adapt a plan when few decisions need to be taken (e.g. in case of temporal failures that do not require the insertion of new actions. . .) and when the elaboration of a complete plan is expensive (cf. the time needed to compute the initial plan in the rover domain). Still, the performances of the plan repair process could be greatly improved if past timepoints and constraints could be removed from the STN (without making the timepoint execution process too expensive).

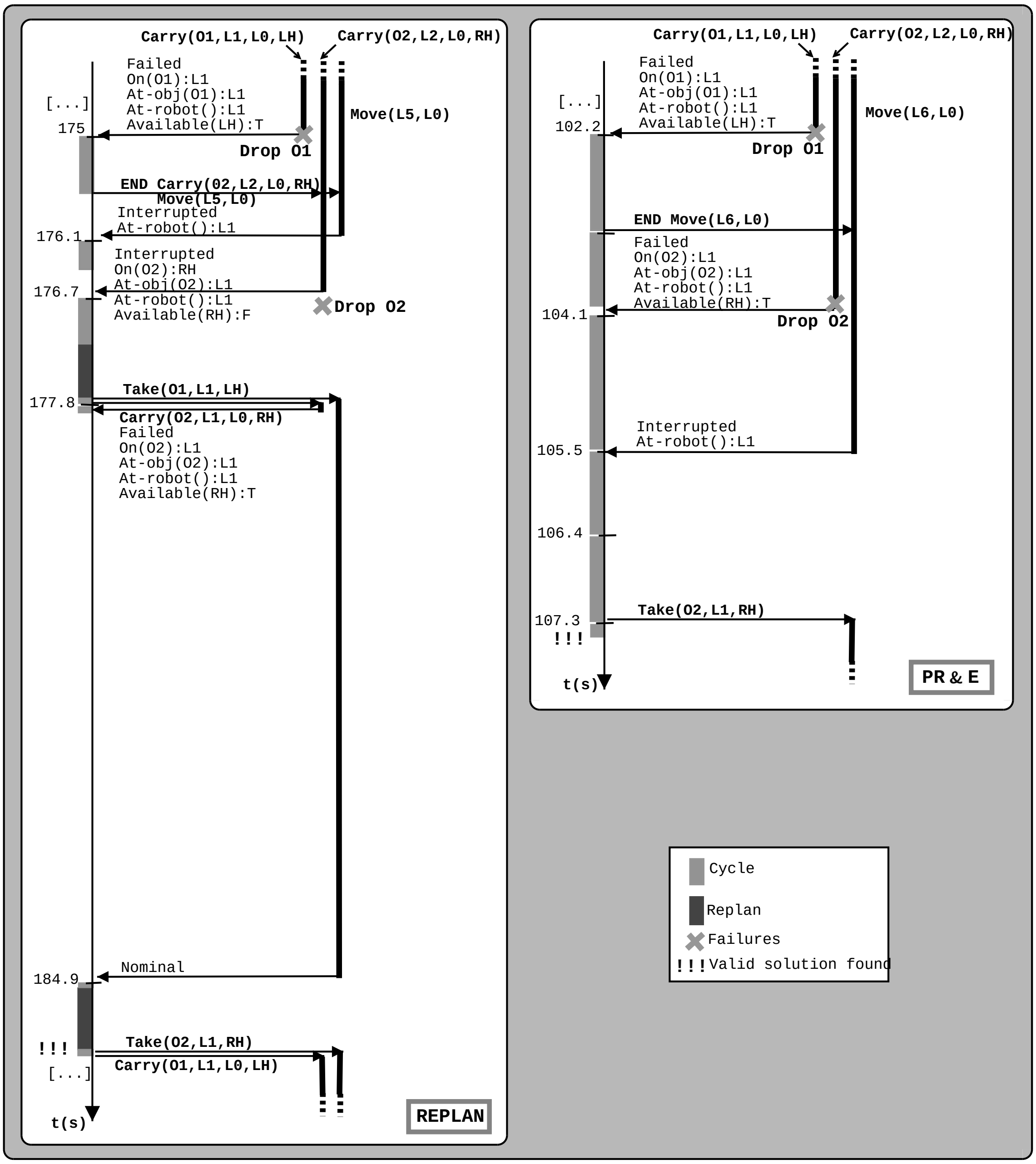


Figure 5.2: *Reaction to a sequence of failures*

5.3 Resources

The domain model presented in the previous sections is modified to take into account the energy consumed by the robot. The energy is represented as a reservoir resource (battery). The main consuming actions are: move, take and put. The quantity consumed by each action is specified according to the duration of the action and a discharge rate. The robot can recharge its battery in L_0. A recharge() action is added to the model. The final quantity produced by this action depends on its duration and a recharge rate.

The decrease of the energy level is simulated in OpenPRS with two discharge rates: a low and permanent one, and a higher one during the execution of the main consuming actions. Furthermore, a procedure checks regularly that there is enough energy to go back to L_0 according to the current simulated position of the robot. The currently running actions are stopped if this safety condition does not hold.

Figure 5.3 shows the evolution of the theoretical energy level (minimal and maximal bounds represented respectively by crosses and circles) and of the simulated level (represented by squares) during the execution of the exploration mission (no object present in the area).

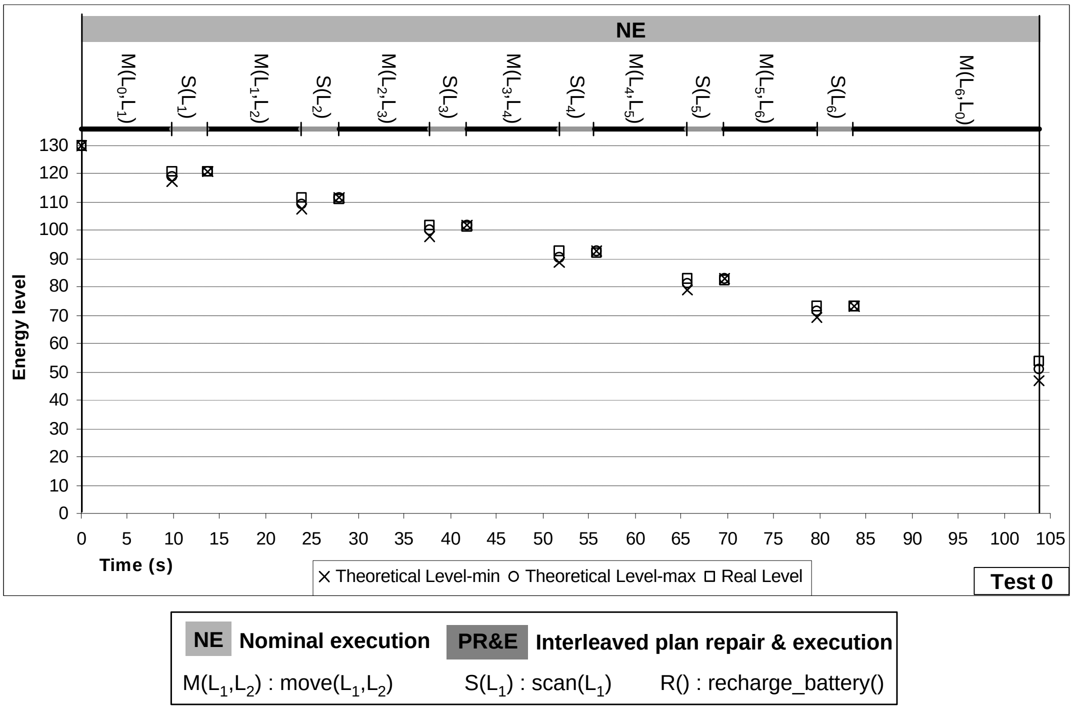


Figure 5.3: *Evolution of the energy level during a nominal execution*

What works...

One of the aims of the IxTeT-eXeC system is to update regularly its knowledge of the resource level, to check the resource usage over the complete mission and to detect potential future resource contentions.

We simulate a "leak" of energy l while the robot is moving from L_1 to L_2. Figures 5.4 and 5.5 represent the evolution of the theoretical and simulated energy levels when the leak is respectively $l = 35$ and $l = 55$. The theoretical level is updated at the end of the move(L_1,L_2) action and the system checks if the plan now contains a resource conflict, i.e. checks if the maximal cliques in the Possible Intersection Graph are over-consuming. In the first case, the leak does not endanger the execution of the remaining actions in the plan. A nominal execution is pursued. In the second case however, the energy left is not sufficient to achieve the plan which is repaired. The actions move(L_2,L_0) and recharge() are added and executed.

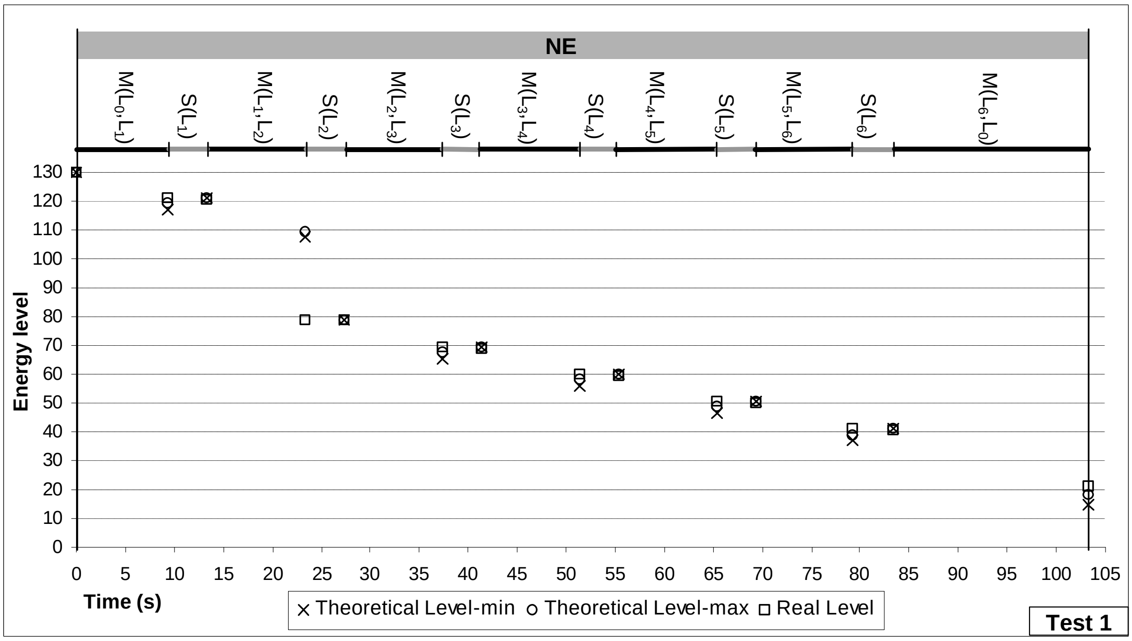


Figure 5.4: *Evolution of the energy level* - $l = 35$

Figure 5.6 presents a case where the capacity is permanently damaged at t=12s (the capacity drops from 130 to 100). The leak $l = 35$ is then simulated as before. This time a conflict is detected and the plan is repaired with the insertion of the actions move(L_2,L_0) and recharge() (the recharge duration takes into account the altered capacity).

The conflict detection is efficient. As an example, the average duration of the resource update (including the level correction and the conflict detection for both ENERGY and ENERGY_CPT resource attributes) during this last run is 131ms.

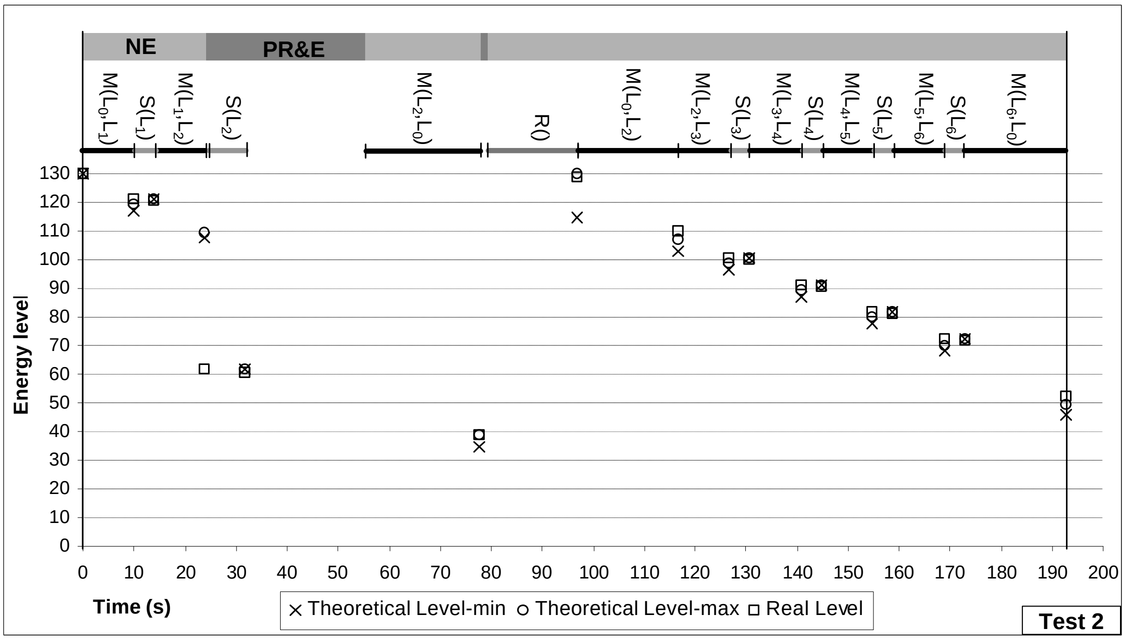


Figure 5.5: *Evolution of the energy level* - $l = 55$

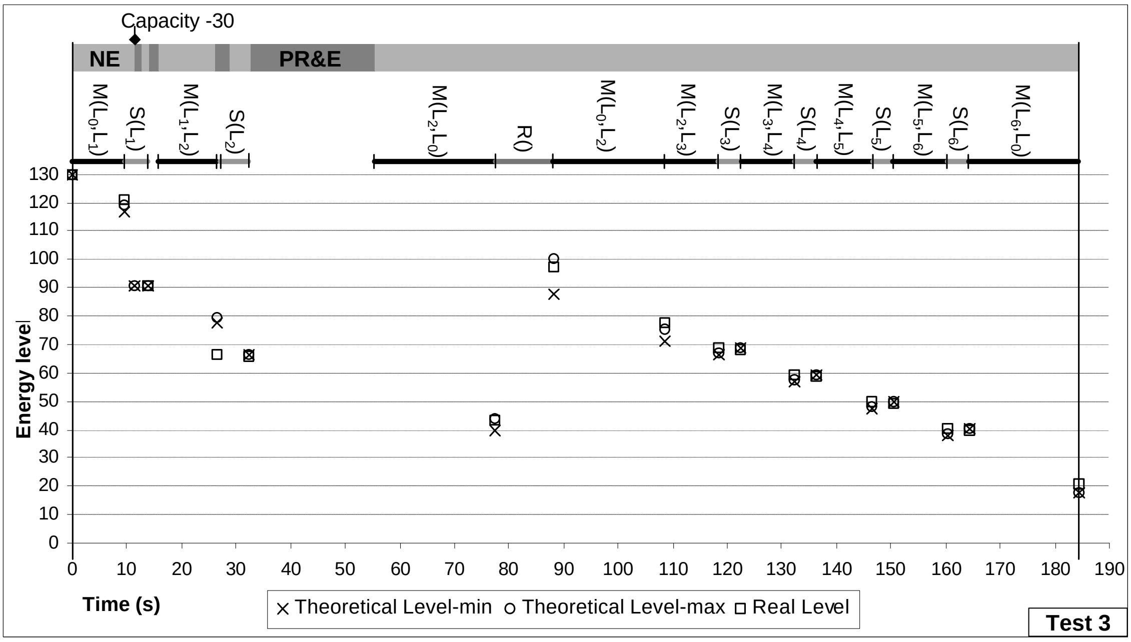


Figure 5.6: *Evolution of the energy level* - $\Delta_{capa} = 30$, $l = 35$

What does not work...

We consider now that the objects O_1 and O_2 are present in the area. The plan is extended by the plan repair process after the insertion of each new goal. The second object induces one resource conflict that can be solved by a *quantity limitation* resolver. However, the actions inserted to handle a third object require a recharge action.

We present 4 tests with the following sets of objects present in the area: $\{O_1,O_2,O_3\}$, $\{O_1,O_2,O_4\}$, $\{O_1,O_2,O_5\}$ or $\{O_1,O_2,O_6\}$. The planning duration explodes when the third goal is inserted. Most of the time is spent in the analysis of the partial plan at each planning step, and more particularly in the analysis of the resource conflicts. This process looks for the set of smallest *Minimal Critical Sets* and computes the associated resolvers (cf. sections 2.3.3 and 2.4.2).

Figure 5.7 represents the duration of the analysis of the conflicts on one resource according to the number of *use* propositions present in the plan. Measures have been done during the plan repair phases that occured in the 4 tests after the insertion of an object-goal. Each point corresponds to the detection of a set of n MCS of size s.[2] The corresponding (s, n) pairs are given for the conflicts induced by the insertion of the object-goal O_3.

The analysis duration explodes when the number of propositions present in the plan increases. We encounter here the problem mentioned page 72. The enumeration method used to compute the set of MCS is in $O(2^k)$ where k is bounded by the size of the maximum clique of the PIG. In our case, the maximal clique is almost equal to the complete set of propositions. In the tests, the plan repair is tractable if the third object is O_6 (and intractable if the object is O_3) thanks to the mechanism of TEXEC that aggregates past *use* propositions in an equivalent one.

Finally, the method used in IXTET for the resource analysis is efficient for certain types of resources: those which lead to maximal cliques of limited size (up to 8). The power or CPU resources, which authorize only a limited number of processes to share the resource, can be handled this way. But the method is inadequate for energy like resources, for which a small quantity of the resource is consumed by each action in the plan.

[2] If a same set of conflicts is detected several times during the plan search, the value in the graph corresponds to the average duration.

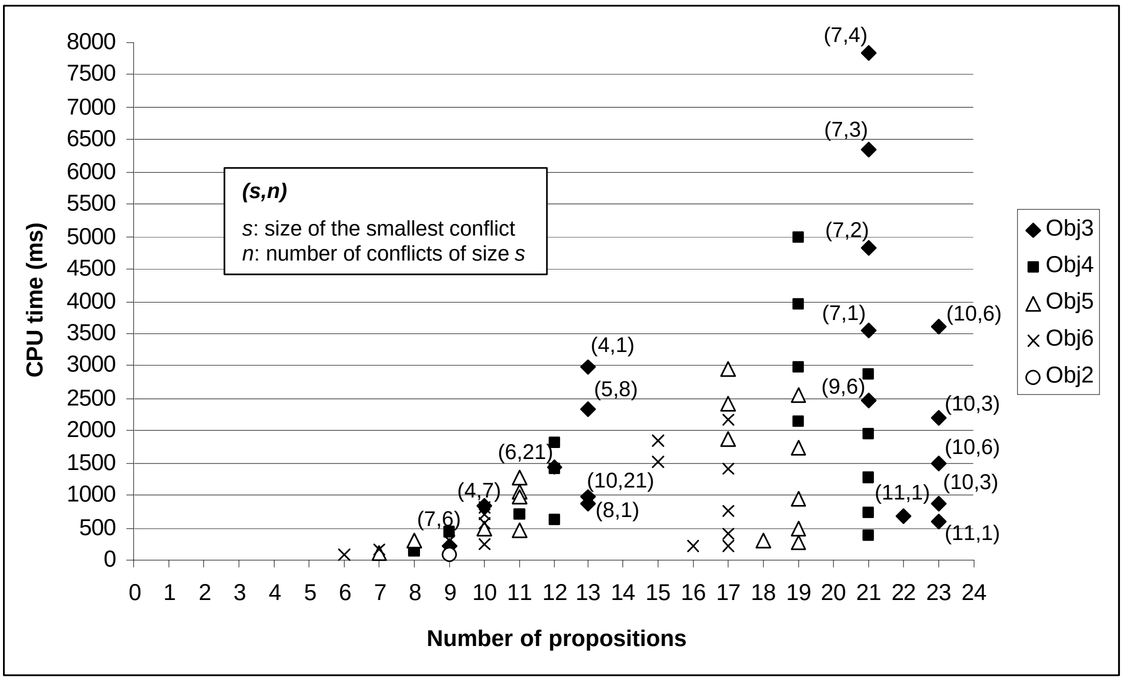


Figure 5.7: *Evolution of the resource analysis duration according to the number of use propositions in the plan*

Conclusion and prospects

The global objective of this work was to integrate mission planning in an autonomous system (e.g. a rover), taking into account the interactions between planning and plan execution; and this in the context of applications involving temporal constraints (goal deadlines) and the management of limited resources.

We have proposed a new framework to integrate deliberative planning, execution control and the reactive adaptation of a plan. These processes are embedded thanks to two components:

- a *planning system* that represents and reasons about time and resources, and with the following properties:
  - a temporal representation based on state variables,
  - a Partial Order Causal Link planning process,
  - it generates flexible plans based on CSP managers (particularly, the time-map relies on a Simple Temporal Network).
- a *temporal executive* that:
  - controls the temporal network of the plan to decide the execution of actions,
  - integrates the system state reports (including resource levels) in the plan,
  - reacts to execution events (action failure, temporal failure, future resource contention, new goal),
  - controls the processes of plan adaptation.

We use the principles and algorithms of the IxTeT planning system, modified to take into account the current execution time and context and the planning duration. We have also improved the expressiveness and flexibility of its resource representation (notably by allowing the usage of variable quantites of a resource). Thus IxTeT provides an initial plan with temporal and resource flexibility which is more robust to execution contingencies.

This plan is then run by the temporal executive following a cycle: integrate external messages, repair the plan if needed, decide which actions to execute. This cycle enables interactions with the controlled system by taking into account runtime failures and timeouts, and updating the plan accordingly.

Two mechanisms of plan adaptation have been implemented:

1. Interleaved plan repair and plan execution.
2. Execution abortion and replanning from scratch.

The first one was motivated by the the fact that some parts of the plan may remain valid and executable and that the plan is temporally flexible, thus allowing postponing and inserting actions. The second one provides a planning search process which is guaranteed to find a solution plan (providing there is enough time before the end of the planning horizon).

This approach has been implemented and integrated in the decisional level of the LAAS architecture, in interaction with an executive based on a procedural reasoning system.

The ability of the system to respect goal deadlines, to react to a series of events, to integrate new goals and to detect and react to future resource conflicts has been shown through tests performed on-board an outdoor robot with an exploration mission and in simulation.

Throughout this document, we have highlighted some limits and prospects:

- **Planning system**
  Non-linear planning has some interesting properties with respect to the interleaving of plan execution and plan adaptation, but its performances limit the size of the planning problems that can be handled (cf. page 80). Future work could take advantage of the heuristic proposed by R. Trinquart to guide the search in the partial plan space [Trinquart 04]. We have also seen (page 171) that the resource management in IxTeT is not well adapted for certain types of resources.

- **Integrating planning and execution**
  The plans produced by IxTeT contain different types of timepoints (start/ end/ intermediate timepoints of actions, goals, contingent events, etc.). The current implementation defines an execution strategy for a restricted set of categories. A future work direction could be to improve the skills of the temporal executive by extending the set of timepoints taken into account during execution. This could be used for instance to check the occurrence of contingent events, to monitor the state of the system during the actions, etc.

  The current policy followed by IxTeT-eXeC to choose between plan repair and replanning is to try plan repair as long as an *executable* plan is available, and then replan with the potential abandon of some goals. Executability is a necessary condition on the plan to perform interleaved plan repair and plan execution. Further work could define more precisely under which conditions a plan repair is likely to fail (cf. page 131). A similar failure and plan analysis could help define more appropriate criteria to select a goal to abandon.

  We have also discussed the interest of taking into account the non controllability of duration links in the propagation algorithms (cf. section 3.8.2). The enforcement of the pseudo and dynamic controllability during the planning and execution processes will reduce the need for plan repair (some work is in progress to address this issue). We have also seen that propagation algorithms in the STN have an impact on the performances of plan repair (cf. page 165), and dynamic controllability can make these algorithms more expensive. The possibility to remove past timepoints should also be considered.

- **Integration in the LAAS architecture for an autonomous robot**

Future work should integrate the Requests Control Level and take into account its reports when requests are killed or rejected. Another important step towards an autonomous behavior consists in defining and implementing potential recovery actions for each error diagnosed and reported by the functional modules.

Conclusion et perspectives

L'objectif principal de ces travaux était d'intégrer la planification de tâches au niveau mission dans un système autonome (par exemple un rover) en tenant compte des interactions entre la planification et l'exécution des plans; et ce pour des applications impliquant des contraintes temporelles et la gestion de ressources limitées.

Nous avons proposé une nouvelle approche pour intégrer l'élaboration de plans, le contrôle de leur exécution et leur adaptation réactive. Ces processus sont assurés par deux composants :

- un *système de planification* qui permet de représenter et raisonner sur le temps et les ressources, et qui présente les propriétés suivantes :
  - une représentation temporelle basée sur des variables d'état,
  - un processus de planification non linéaire causal,
  - il génère des plans flexibles basés des CSPs (en particulier, le réseau temporel est un STN).
- un *exécutif temporel* qui :
  - contrôle le réseau temporel du plan et décide de l'exécution des actions,
  - intègre dans le plan les bilans sur l'état du système (notamment le niveau des ressources),
  - réagit aux événements survenant pendant l'exécution (échec d'une action, échec temporel, conflit de ressource, nouveau but),
  - contrôle les processus d'adaptation du plan.

Nous nous sommes basés sur les principes et algorithmes du planificateur IxTeT, que nous avons modifié pour prendre en compte le contexte d'exécution (entre autres l'instant courant) et la durée de la planification. Nous avons également amélioré l'expressivité et la flexibilité de la représentation des ressources dans IxTeT (en permettant notamment de spécifier l'utilisation de quantités variables). Ainsi, un plan initial produit par IxTeT est flexible temporellement et au niveau des ressources et sera plus robuste aux aléas de l'exécution.

Ce plan est "déroulé" par l'exécutif temporel selon un cycle : intégrer les messages externes, réparer le plan si nécessaire, décider des actions à exécuter. Ce cycle permet d'interagir avec le système contrôlé, de prendre en compte les échecs et les dépassements de délai en cours d'exécution et d'adapter le plan en conséquence.

Deux mécanismes d'adaptation du plan ont été implémentés :

1. Réparation et exécution de plan entrelacés,
2. Arrêt de l'exécution et replanification complète.

Le premier mécanisme est justifié par le fait que certaines parties du plan peuvent rester valides et exécutables et que le plan est temporellement flexible, ce qui permet de retarder l'exécution des actions et d'en insérer de nouvelles. Le deuxième mécanisme fournit un processus de planification complet qui offre la garantie de trouver un plan solution (à condition qu'il reste suffisamment de temps avant la fin de l'horizon de planification).

Cette approche a été implémentée et intégrée dans le niveau décisionnel de l'architecture LAAS, en interaction avec un autre exécutif basé sur un système de raisonnement procédural.

La capacité du système à respecter les dates limites des buts, à réagir à une série d'événements, à intégrer de nouveaux buts et à detecter et réparer des conflits de ressource futurs a été montrée à la fois en simulation et à travers des tests menés à bord d'un robot d'extérieur avec une mission d'exploration.

Tout au long de ce document, nous avons mis en avant certaines limites de l'approche et de futures pistes de recherche :

- **Système de planification**
  La planification non linéaire a des propriétés intéressantes concernant l'entrelacement de l'exécution et de l'adaptation d'un plan, mais ses performances limitent la taille des problèmes de planification qui peuvent être traités (cf. page 80). Il pourrait être intéressant d'exploiter l'heuristique proposée par R. Trinquart pour guider la recherche dans l'espace des plans partiels [Trinquart 04]. Nous avons également constaté (page 171) que la gestion des ressources dans IxTeT n'était pas bien adaptée pour certaines catégories de ressource.

- **Intégrer planification et exécution**
  Les plans produits par IxTeT comprennent différents types de variables d'instant (instants de début/ de fin/ intermédiaires d'une action, de buts, d'événements contingents, etc.). L'implémentation actuelle ne définit une stratégie d'exécution que pour un nombre restreint de ces catégories. Un futur axe de travail pourrait étendre les capacités de l'exécutif temporel en tenant en compte d'autres types d'instants durant l'exécution, par exemple pour vérifier la réalisation d'un événement contingent, pour surveiller l'état du système au cours de l'exécution des actions, etc.

  La stratégie suivie actuellement par IxTeT-eXeC pour choisir entre réparation de plan et replanification est de tenter une réparation tant qu'un plan *exécutable* est disponible, puis de replanifier avec potentiellement l'abandon de certains buts.
  L'exécutabilité est une condition nécessaire sur le plan pour pouvoir effectuer en parallèle sa réparation et son exécution. Il serait intéressant de définir plus précisément sous quelles conditions une réparation de plan est susceptible d'échouer (cf. page 131). Une analyse similaire du plan et des cas d'échec pourrait également permettre de définir de meilleurs critères de sélection des buts à abandonner.

  Nous avons également évoqué l'intérêt de prendre en compte la non contrôlabilité des durées dans les algorithmes de propagation des contraintes (cf. section 3.8.2). Le respect des propriétes de pseudo-contrôlabilité et de contrôlabilité dynamique pendant les processus de planification et d'exécution pourrait permettre de réduire les cas nécessitant une réparation (des travaux sont en cours dans ce sens). Nous avons également vu que les algorithmes de propagation pour le STN ont un impact sur les performances de la réparation de plan (cf. page 165), et que rechercher la contrôlabilité dynamique peut rendre ces algorithmes plus coûteux. La possibilité de supprimer les variables d'instant révolues devrait aussi être étudiée.

- **Intégration dans l'architecture LAAS**
  Il faut encore intégrer le niveau de Contrôle des Requêtes et prendre en compte les bilans qu'il renvoie lors du rejet de requêtes. Il reste enfin une étape importante pour parvenir à un comportement réellement autonome : la définition et l'implémentation d'actions de recouvrement pour chaque erreur pouvant être diagnostiquée et retournée par les modules du niveau fonctionnel.

Appendix 1

Algorithms for maximal cliques computation

A clique in an unoriented graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E. A maximal clique is a clique that is not contained in any larger clique.

The following algorithms compute S_M, the set of maximal cliques containing the clique $\mathcal{C} = \{x_1, \ldots, x_k\}$. The neighbourhood of a vertex x is noted $N(x) = \{y | y \neq x, (x, y) \in E\}$.

```
**Compute_Maximal_Cliques($\mathcal{C}$)**

$S_M \leftarrow \emptyset$;
$Partition \leftarrow \emptyset$;
$MutualN \leftarrow N(x_1)$;
for i:=2 to $k$ do
  $MutualN \leftarrow MutualN \cap N(x_i)$;
if (Is_Clique($MutualN$, $Partition$)) do
  % There is only one maximal clique containing $\mathcal{C}$
  $S_M \leftarrow (\mathcal{C} \cup MutualN)$;
else
  % Several maximal cliques
  $S_M \leftarrow$ Partition_Mutual_Neighbours($\mathcal{C}$, $MutualN$, $Partition$);
return $S_M$;
```

```
**Is_Clique**($MutualN$, $Partition$)
% Returns true if $MutualN$ is a clique.
% $MutualN = \{y_1, \ldots, y_l\}$

for i:=1 to $l$ do
  for j:=i+1 to $l$ do
    if ($y_i \notin N(y_j)$) do
      $Partition \leftarrow Partition \cup \{y_i, y_j\}$;
return ($Partition == \emptyset$);
```

```
**Partition_Mutual_Neighbours**($\mathcal{C}$, $MutualN$, $Partition$)

$localS_M \leftarrow \emptyset$;
while ($Partition \neq \emptyset$) do
  $localPartition \leftarrow \emptyset$;
  get first element $e$ in $Partition$;
  remove $e$ from $Partition$;
  $localMutualN \leftarrow MutualN \cap N(e)$;
  if (Is_Clique($localMutualN$, $localPartition$)) do
    $localS_M \leftarrow localS_M \cup (\mathcal{C} \cup localMutualN \cup \{e\})$;
    $Partition \leftarrow Partition - localMutualN$;
  else
    $Partition \leftarrow Partition - localPartition$;
    $localS_M \leftarrow localS_M \cup$ Partition_Mutual_Neighbours($\mathcal{C} \cup \{e\}$, $localMutualN$, $localPartition$);
return $localS_M$;
```

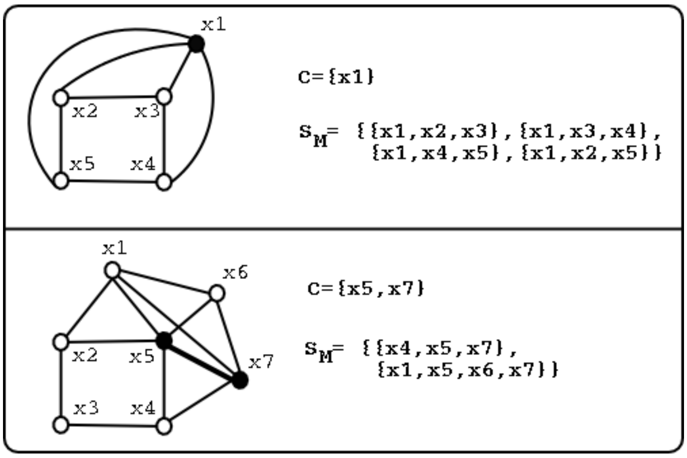


Figure 1.1: *Examples*

Appendix 2

Tests on Dala

2.1 IxTeT-eXeC model

An IxTeT model is specified through three files.

1. The file “model-relations” contains the definition of the constants and attributes (logical, resource, rigid).
2. The file “model.task” contains the model of actions.
3. The file “model-init.task” contains the description of the initial situation, initial goals and profiles of contingent attributes.

These files are compiled and loaded in IxTeT-eXeC at the initialization of the mission.

“model-relations”

```
constant BOOL = {T, F};
constant OBJECTS = {OBJ1, OBJ2, OBJ3, OBJ4, OBJ5, OBJ6};
constant STATUS = {NONE,DONE};
constant PANTILT_POSITIONS = {FORWARD, DOWNWARD, OTHER_POSITION};
constant VISIBILITY_WINDOWS = {W1,W2};
constant IN_OUT = {IN,OUT};

attribute VISIBILITY_WINDOW(?w){
        ?w in VISIBILITY_WINDOWS;
        ?value in IN_OUT;
}

attribute COMMUNICATION(?w){
        ?w in VISIBILITY_WINDOWS;
        ?value in STATUS | {COMMUNICATION_IDLE};
}

attribute PTU_DRIVER_INITIALIZED(){
        ?value in BOOL | {PTU_DRIVER_INITIALIZED_IDLE};
}

attribute MVT_GENERATION_INITIALIZED(){
        ?value in BOOL | {MVT_GENERATION_INITIALIZED_IDLE};
}
```

```
attribute AT_ROBOT_X() {
        ?value in ]-oo,+oo[;
}

attribute AT_ROBOT_Y() {
        ?value in ]-oo,+oo[;
}

attribute ROBOT_STATUS(){
        ?value in {MOVING,STILL};
}

attribute PICTURE(?o, ?x, ?y) {
        ?o in OBJECTS;
        ?x in ]-oo,+oo[;
        ?y in ]-oo,+oo[;
        ?value in STATUS | {PICTURE_IDLE};
}

attribute PAN_TILT_UNIT_POSITION() {
        ?value in PANTILT_POSITIONS | {PAN_TILT_UNIT_POSITION_IDLE};
}

resource CAMERA() {
        defaultcapacity = 1.;
}

resource CHANNEL() {
        defaultcapacity = 1.;
}

reservoir resource STORAGE(){
        defaultcapacity = 66000.;
}

// The distance between two points is estimated by d = X+Y, with
// X = |xf-xi| et Y=|yf-yi|
#define distance(_xi, _yi, _xf, _yf, _d)\
  variable ?Y;\
  variable ?Yc;\
  variable ?Ypos;\
  variable ?X;\
  variable ?Xc;\
  variable ?Xpos;\
  _xi in ]-oo,+oo[;\
  _xf in ]-oo,+oo[;\
  _yi in ]-oo,+oo[;\
  _yf in ]-oo,+oo[;\
  _d in ]-oo,+oo[;\
  ?X in ]-oo,+oo[;\
  ?Xc in ]-oo,+oo[;\
  ?Xpos in ]-oo,+oo[;\
  ?Y in ]-oo,+oo[;\
  ?Yc in ]-oo,+oo[;\
  ?Ypos in ]-oo,+oo[;\
  ?X =. _xf -. _xi;\
  ?Xc =. _xi -. _xf;\
  ?Y =. _yf -. _yi;\
  ?Yc =. _yi -. _yf;\
  ?Xpos =. max(?X,?Xc);\
  ?Ypos =. max(?Y,?Yc);\
  _d =. ?Xpos +. ?Ypos
```

```
// The above approximation of the distance is improved by d*[0.7,1]
#define distance_uncertainty(_u)\
  _u in [0.7,1]

#define speed(_sp)\
  _sp in [0.03,0.1]

#define compression_rate(_cr)\
  _cr in [0.1,0.13]

#define download_rate(_r)\
  _r in [800,800]

#define storage_capacity(_cs)\
  _cs in [66000,66000]
```

“model.task”

```
#include "model-relations"

task MOVE(?x1,?y1,?x2,?y2)(t_start, t_end){
    ?x1 in ]-oo,+oo[;
    ?x2 in ]-oo,+oo[;
    ?y1 in ]-oo,+oo[;
    ?y2 in ]-oo,+oo[;

    hold(PAN_TILT_UNIT_POSITION():FORWARD,(t_start,t_end));
    hold(MVT_GENERATION_INITIALIZED():T,(t_start,t_end));
    event(AT_ROBOT_X():(?x1,1000),t_start);
    hold(AT_ROBOT_X():1000,(t_start,t_end));
    event(AT_ROBOT_X():(1000,?x2),t_end);
    event(AT_ROBOT_Y():(?y1,1000),t_start);
    hold(AT_ROBOT_Y():1000,(t_start,t_end));
    event(AT_ROBOT_Y():(1000,?y2),t_end);
    event(ROBOT_STATUS():(STILL,MOVING),t_start);
    hold(ROBOT_STATUS():MOVING,(t_start,t_end));
    event(ROBOT_STATUS():(MOVING,STILL),t_end);

    use(CAMERA():1,(t_start,t_end));

    variable ?di;
    distance(?x1,?y1,?x2,?y2,?di);
    variable ?du;
    distance_uncertainty(?du);
    variable ?dist;
    ?dist =. ?di *. ?du;
    variable ?s;
    speed(?s);
    variable ?duration;
    ?dist =. ?s *. ?duration;
    ?duration =. t_end - t_start;
}latePreemptive

task MOVEX(?x1,?y1,?x2,?y2)(t_start, t_end){
    ?x1 in ]-oo,+oo[;
    ?x2 in ]-oo,+oo[;
    ?y1 in ]-oo,+oo[;
    ?y2 in ]-oo,+oo[;
    ?y1 = ?y2;
    hold(PAN_TILT_UNIT_POSITION():FORWARD,(t_start,t_end));
    hold(MVT_GENERATION_INITIALIZED():T,(t_start,t_end));
    event(AT_ROBOT_X():(?x1,1000),t_start);
    hold(AT_ROBOT_X():1000,(t_start,t_end));
    event(AT_ROBOT_X():(1000,?x2),t_end);
    hold(AT_ROBOT_Y():?y2,(t_start,t_end));
    event(ROBOT_STATUS():(STILL,MOVING),t_start);
    hold(ROBOT_STATUS():MOVING,(t_start,t_end));
    event(ROBOT_STATUS():(MOVING,STILL),t_end);

    use(CAMERA():1,(t_start,t_end));

    variable ?di;
    distance(?x1,?y1,?x2,?y2,?di);
    variable ?du;
    distance_uncertainty(?du);
    variable ?dist;
    ?dist =. ?di *. ?du;
    variable ?s;
    speed(?s);
    variable ?duration;
    ?dist =. ?s *. ?duration;
    ?duration =. t_end - t_start;
}latePreemptive


task MOVEY(?x1,?y1,?x2,?y2)(t_start, t_end){
    ?x1 in ]-oo,+oo[;
    ?x2 in ]-oo,+oo[;
    ?y1 in ]-oo,+oo[;
    ?y2 in ]-oo,+oo[;
    ?x1 = ?x2;

    hold(PAN_TILT_UNIT_POSITION():FORWARD,(t_start,t_end));
    hold(MVT_GENERATION_INITIALIZED():T,(t_start,t_end));
    event(AT_ROBOT_Y():(?y1,1000),t_start);
    hold(AT_ROBOT_Y():1000,(t_start,t_end));
    event(AT_ROBOT_Y():(1000,?y2),t_end);
    hold(AT_ROBOT_X():?x2,(t_start,t_end));
    event(ROBOT_STATUS():(STILL,MOVING),t_start);
    hold(ROBOT_STATUS():MOVING,(t_start,t_end));
    event(ROBOT_STATUS():(MOVING,STILL),t_end);

    use(CAMERA():1,(t_start,t_end));

    variable ?di;
    distance(?x1,?y1,?x2,?y2,?di);
    variable ?du;
    distance_uncertainty(?du);
    variable ?dist;
    ?dist =. ?di *. ?du;
    variable ?s;
    speed(?s);
    variable ?duration;
    ?dist =. ?s *. ?duration;
    ?duration =. t_end - t_start;
}latePreemptive


task MOVE_PAN_TILT_UNIT(?initpos,?finpos)(t_start, t_end){
    ?initpos in PANTILT_POSITIONS;
    ?finpos in PANTILT_POSITIONS;

    event(PAN_TILT_UNIT_POSITION():(?initpos,PAN_TILT_UNIT_POSITION_IDLE),t_start);
    hold(PAN_TILT_UNIT_POSITION():PAN_TILT_UNIT_POSITION_IDLE,(t_start,t_end));
    event(PAN_TILT_UNIT_POSITION():(PAN_TILT_UNIT_POSITION_IDLE,?finpos),t_end);
    hold(PTU_DRIVER_INITIALIZED():T,(t_start,t_end));

    (t_end - t_start) in [3,10];
}latePreemptive


task TAKE_PICTURE(?obj,?x,?y)(t_start, t_end){
    ?obj in OBJECTS;
    ?x in ]-oo,+oo[;
    ?y in ]-oo,+oo[;

    hold(AT_ROBOT_X():?x,(t_start,t_end));
    hold(AT_ROBOT_Y():?y,(t_start,t_end));
    hold(PAN_TILT_UNIT_POSITION():DOWNWARD,(t_start,t_end));
    event(PICTURE(?obj,?x,?y):(NONE,PICTURE_IDLE),t_start);
    hold(PICTURE(?obj,?x,?y):PICTURE_IDLE,(t_start,t_end));
    event(PICTURE(?obj,?x,?y):(PICTURE_IDLE,DONE),t_end);

    use(CAMERA(): 1, (t_start,t_end));

    variable ?image_size;
    variable ?cr;
    compression_rate(?cr);
    ?image_size =. 175610 *. ?cr;
    consume(STORAGE():?image_size,t_start);

    (t_end - t_start) in ]2,60];
}nonPreemptive


task DOWNLOAD_IMAGES()(t_start,t_end){
       use(CAMERA(): 1, (t_start,t_end));
       use(CHANNEL(): 1, (t_start,t_end));

       variable ?q;
       variable ?qbis;
       variable ?duration;
       variable ?capa;
       storage_capacity(?capa);
       variable ?rate;
       download_rate(?rate);
       ?qbis =. ?capa -. ?q;
       ?qbis in [0,+oo[;
       ?q =. ?duration *. ?rate;
       produce(STORAGE():?q,t_end);
       ?duration =. t_end - t_start;
}latePreemptive


task COMMUNICATE(?w)(t_start, t_end){
    ?w in VISIBILITY_WINDOWS;

    hold(VISIBILITY_WINDOW(?w):IN,(t_start,t_end));
    event(COMMUNICATION(?w):(NONE,COMMUNICATION_IDLE),t_start);
    hold(COMMUNICATION(?w):COMMUNICATION_IDLE,(t_start,t_end));
    event(COMMUNICATION(?w):(COMMUNICATION_IDLE,DONE),t_end);
    hold(ROBOT_STATUS():STILL,(t_start,t_end));
    use(CHANNEL(): 1, (t_start,t_end));

    (t_end - t_start) in ]100,200];
}latePreemptive


//- Failure recovery actions
task INIT_PTU_DRIVER()(t_start, t_end){
    // Do nothing while the modules are initialized
    hold(ROBOT_STATUS():STILL,(t_start,t_end));
    use(CAMERA(): 1, (t_start,t_end));

    event(PTU_DRIVER_INITIALIZED():(F,PTU_DRIVER_INITIALIZED_IDLE),t_start);
    hold(PTU_DRIVER_INITIALIZED():PTU_DRIVER_INITIALIZED_IDLE,(t_start,t_end));
    event(PTU_DRIVER_INITIALIZED():(PTU_DRIVER_INITIALIZED_IDLE,T),t_end);
    (t_end - t_start) in ]2,60];
}nonPreemptive


task INIT_MVT_GENERATION()(t_start, t_end){
    // Do nothing while the modules are initialized
    hold(ROBOT_STATUS():STILL,(t_start,t_end));
    use(CAMERA(): 1, (t_start,t_end));

    event(MVT_GENERATION_INITIALIZED():(F,MVT_GENERATION_INITIALIZED_IDLE),t_start);
    hold(MVT_GENERATION_INITIALIZED():MVT_GENERATION_INITIALIZED_IDLE,(t_start,t_end));
    event(MVT_GENERATION_INITIALIZED():(MVT_GENERATION_INITIALIZED_IDLE,T),t_end);
    (t_end - t_start) in ]2,60];
}nonPreemptive
```

“model-init.task”

```
#include "model-relations"

task Init()(t_start,t_end){

   timepoint t_svisi1, t_evisi1;
   timepoint t_svisi2, t_evisi2;

   timepoint t_goal1;
   timepoint t_goal3s,t_goal3e;
   timepoint t_goal4s,t_goal4e;
   timepoint t_goal5s,t_goal5e;
   timepoint t_goal6s,t_goal6e;
   timepoint t_goal7s, t_goal7e;
   timepoint t_goal8s, t_goal8e;

    //initial situation
    explained event(AT_ROBOT_X():(?,0),t_start);
    explained event(AT_ROBOT_Y():(?,0),t_start);
    explained event(ROBOT_STATUS():(?,STILL),t_start);
    explained event(PAN_TILT_UNIT_POSITION():(?,FORWARD),t_start);
    explained event(PTU_DRIVER_INITIALIZED():(?,T),t_start);
    explained event(MVT_GENERATION_INITIALIZED():(?,T),t_start);
    explained event(COMMUNICATION(W1):(?,NONE),t_start);
    explained event(COMMUNICATION(W2):(?,NONE),t_start);
    contingent event(VISIBILITY_WINDOW(W1):(?,OUT),t_start);
    contingent event(VISIBILITY_WINDOW(W2):(?,OUT),t_start);

    variable ?x1,?y1;
    ?x1 in ]-oo,+oo[;
    ?y1 in ]-oo,+oo[;
    explained event(PICTURE(OBJ1,?x1,?y1):(?,NONE),t_start);
    variable ?x2,?y2;
    ?x2 in ]-oo,+oo[;
    ?y2 in ]-oo,+oo[;
    explained event(PICTURE(OBJ2,?x2,?y2):(?,NONE),t_start);
    variable ?x3,?y3;
    ?x3 in ]-oo,+oo[;
    ?y3 in ]-oo,+oo[;
    explained event(PICTURE(OBJ3,?x3,?y3):(?,NONE),t_start);
    variable ?x4,?y4;
    ?x4 in ]-oo,+oo[;
    ?y4 in ]-oo,+oo[;
    explained event(PICTURE(OBJ4,?x4,?y4):(?,NONE),t_start);
    variable ?x5,?y5;
    ?x5 in ]-oo,+oo[;
    ?y5 in ]-oo,+oo[;
    explained event(PICTURE(OBJ5,?x5,?y5):(?,NONE),t_start);
    variable ?x6,?y6;
    ?x6 in ]-oo,+oo[;
    ?y6 in ]-oo,+oo[;
    explained event(PICTURE(OBJ6,?x6,?y6):(?,NONE),t_start);

    //- Visibility windows
    contingent event(VISIBILITY_WINDOW(W1):(OUT,IN),t_svisi1);
    contingent event(VISIBILITY_WINDOW(W1):(IN,OUT),t_evisi1);

    contingent event(VISIBILITY_WINDOW(W2):(OUT,IN),t_svisi2);
    contingent event(VISIBILITY_WINDOW(W2):(IN,OUT),t_evisi2);

   (t_evisi1 - t_svisi1) in [120,121];
   (t_svisi1 - t_start) in [300,301];

   (t_evisi2 - t_svisi2) in [120,121];
   (t_svisi2 - t_start) in [600,601];

    //- Goals
    //- Be back near the lander at the end
    hold(AT_ROBOT_X():0,(t_goal1,t_end)) goal(2,0);
    hold(AT_ROBOT_Y():0,(t_goal1,t_end)) goal(2,0);

    //- Communication with ground stations
    hold(COMMUNICATION(W1):DONE,(t_goal3s,t_goal3e)) goal(1,0);
    hold(COMMUNICATION(W2):DONE,(t_goal4s,t_goal4e)) goal(1,0);

    //- Pictures
    hold(PICTURE(OBJ1,0,0):DONE,(t_goal5s,t_goal5e)) goal(1,0);
    hold(PICTURE(OBJ2,9,0):DONE,(t_goal6s,t_goal6e)) goal(1,0);
    hold(PICTURE(OBJ3,8,-5):DONE,(t_goal8s,t_goal8e)) goal(1,0);
    hold(PICTURE(OBJ4,10,-3):DONE,(t_goal7s,t_goal7e)) goal(1,0);

    (t_end - t_goal1) in [2,2];
    t_goal1 >t_goal8s;
    t_goal8s > t_goal7s;
    t_goal7s > t_goal6s;
    t_goal6s > t_goal5s;
    (t_end-t_start) in ]800,2000];
}earlyPreemptive
```

2.2 IxTeT-eXeC trace of Run 3

```
CYCLE N°1 - WAKE UP AT : 0.027
########### ACT #############
Instantiation of Start timepoint 45 at 0.027
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 14 (PARAMETERS FORWARD DOWNWARD))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [46] at 8.027
CYCLE DURATION : 0.123

MsgQueue not empty : [MsgBilan (BILAN, Task 14, nominal, state bilan[] , resource bilan  []), timepoint 46]

CYCLE N°2 - WAKE UP AT : 5.37
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 3.027
Timepoint upper_bound : 10.027
Instantiation of Msg timepoint 46 at 5.37
Update ExecTPs : [25]
########### ACT #############
Instantiation of Start timepoint 25 at 5.38
Send-command : (LAUNCH IXTET_EXEC TAKE_PICTURE 4 (PARAMETERS OBJ1 0 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [26] at 65.38
CYCLE DURATION : 0.081

MsgQueue not empty : [MsgBilan (BILAN, Task 4, nominal, state bilan[] ,
resource bilan  [(CAMERA() 1), (STORAGE_CPT() 21812), (STORAGE() 44188)]), timepoint 26]

CYCLE N°3 - WAKE UP AT : 7.758
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 7.39
Timepoint upper_bound : 65.38
Instantiation of Msg timepoint 26 at 7.758
Update ExecTPs : [47, 14]
########### ACT #############
Instantiation of Start timepoint 47 at 7.768
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 15 (PARAMETERS DOWNWARD FORWARD))
Update ExecTPs : [14]
Instantiation of Goal timepoint 14 at 7.768
Update ExecTPs : [15]
Instantiation of Goal timepoint 15 at 7.778
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [48] at 15.768
CYCLE DURATION : 0.489

MsgQueue not empty : [MsgBilan (BILAN, Task 15, nominal, state bilan[] , resource bilan  []), timepoint 48]

CYCLE N°4 - WAKE UP AT : 11.606
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 10.768
Timepoint upper_bound : 17.768
Instantiation of Msg timepoint 48 at 11.606
Update ExecTPs : [35]
########### ACT #############
Instantiation of Start timepoint 35 at 11.616
Send-command : (LAUNCH IXTET_EXEC MOVEX 9 (PARAMETERS 0 0 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [36] at 292.949
CYCLE DURATION : 0.081

CYCLE N°5 - WAKE UP AT : 292.986
########### ACT #############
Send-command : (END 9)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.052

MsgQueue not empty : [MsgBilan (BILAN, Task 9, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -4.42851), (AT_ROBOT_X() 7.25533), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 36]
```

```
CYCLE N°6 - WAKE UP AT : 297.583
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 71.116
Timepoint upper_bound : 294.949
New timepoint 57
Waiting End Timepoint 57 TIMED OUT
New state insertion...
34 causal links are removed on the attributes:
[MVT_GENERATION_INITIALIZED(), AT_ROBOT_X(), AT_ROBOT_Y(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION()]
Update ExecTPs : [43]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 1
... No solution found
Update ExecTPs : [43]
########### ACT #############
Timepoint 43 is not supported, upper_bound : 589.44
-> WaitingExecTPs
Update ExecTPs : [43]
WaitingExecTPs : [43]
############################
Next ExecTPs [43] at 297.593
CYCLE DURATION : 1.949

CYCLE N°7 - WAKE UP AT : 299.532
########### PLAN REPAIR #############
nb planning steps: 13
... No solution found
Update ExecTPs : [43, 2]
########### ACT #############
Timepoint 43 is not supported, upper_bound : 589.44
-> WaitingExecTPs
Update ExecTPs : [43, 2]
Instantiation of Extern timepoint 2 at 300
Update ExecTPs : [29]
Timepoint 29 is not supported, upper_bound : 320.98
-> WaitingExecTPs
Update ExecTPs : [29]
WaitingExecTPs : [29, 43]
############################
Next ExecTPs [29, 43] at 297.593
CYCLE DURATION : 1.362

CYCLE N°8 - WAKE UP AT : 300.894
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 14
... No solution found
Update ExecTPs : [58, 29]
########### ACT #############
Timepoint 29 is not supported, upper_bound : 320.98
-> WaitingExecTPs
Update ExecTPs : [58, 29]
Timepoint 58 is not supported, upper_bound : 549.718
-> WaitingExecTPs
Update ExecTPs : [58]
WaitingExecTPs : [58, 29]
############################
Next ExecTPs [58, 29] at 300.01
CYCLE DURATION : 1.308

CYCLE N°9 - WAKE UP AT : 302.203
########### PLAN REPAIR #############
nb planning steps: 23
... Solution found
Update ExecTPs : [29]
########### ACT #############
Instantiation of Start timepoint 29 at 302.203
Send-command : (LAUNCH IXTET_EXEC COMMUNICATE 6 (PARAMETERS W1))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [30] at 418.99
CYCLE DURATION : 1.155

MsgQueue not empty : [MsgBilan (BILAN, Task 6, nominal, state bilan[] , resource bilan  [(CHANNEL() 1)]), timepoint 30]

CYCLE N°10 - WAKE UP AT : 408.286
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 402.213
Timepoint upper_bound : 420.99
Instantiation of Msg timepoint 30 at 408.286
Update ExecTPs : [58, 10]
########### ACT #############
Instantiation of Start timepoint 58 at 408.296
Send-command : (LAUNCH IXTET_EXEC MOVE 20 (PARAMETERS 7.255328 -4.428510 9 0))
Update ExecTPs : [10]
Instantiation of Goal timepoint 10 at 408.296
Update ExecTPs : [11]
Instantiation of Goal timepoint 11 at 408.306
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [3] at 420
CYCLE DURATION : 0.293

CYCLE N°11 - WAKE UP AT : 420.007
########### ACT #############
Instantiation of Extern timepoint 3 at 420.007
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [59] at 587.44
CYCLE DURATION : 0.102

CYCLE N°12 - WAKE UP AT : 587.458
########### ACT #############
Send-command : (END 20)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.044

MsgQueue not empty : [MsgBilan (BILAN, Task 20, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -1.70755), (AT_ROBOT_X() 10.1826), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 59]

CYCLE N°13 - WAKE UP AT : 591.196
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 448.008
Timepoint upper_bound : 589.43
New timepoint 60
Waiting End Timepoint 60 TIMED OUT
NO TEMPORAL FLEXIBILITY LEFT, replan
Update ExecTPs : []
########### COMPLETE REPLANNING #############
REPLANNING ATTEMPT N°1
with 5 goals:
  * Priority 2,
    Minimal achievement duration 0s,
    Constraint with origin: ] 798 , 1998],
    Duration: [ 2 , 2 ],
    Proposition: hold(AT_ROBOT_X():0, (6,1))

  * Priority 1,
    Minimal achievement duration 0s,
    Constraint with origin: ] 0 , 1998 [,
    Duration: ] 0 , 2000 [,
    Proposition: hold(PICTURE(OBJ3,8,-5):DONE, (7,8))

  * Priority 1,
    Minimal achievement duration 0s,
    Constraint with origin: ] 0 , 1998 [,
    Duration: ] 0 , 2000 [,
    Proposition: hold(PICTURE(OBJ2,9,0):DONE, (9,10))

  * Priority 1,
    Minimal achievement duration 0s,
    Constraint with origin: [ 0 , 2000 [,
    Duration: ] 0 , 2000 ]
    Proposition: hold(COMMUNICATION(W2):DONE, (11,12))

  * Priority 2,
    Minimal achievement duration 0s,
    Constraint with origin: ] 798 , 1998 ],
    Duration: [ 2 , 2 ]
    Proposition: hold(AT_ROBOT_Y():0, (6,1))
Update replanning_limit_time 1995.98, with current_time : 591.608 and backtrack 0
Update replanning_limit_time 1995.98, with current_time : 591.612 and backtrack 0
Update replanning_limit_time 1995.98, with current_time : 591.623 and backtrack 0
Update replanning_limit_time 1995.98, with current_time : 591.646 and backtrack 0
Update replanning_limit_time 1899.98, with current_time : 591.679 and backtrack 0
Update replanning_limit_time 1899.98, with current_time : 591.704 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.731 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.755 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.784 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.815 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.825 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.835 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.845 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.859 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.868 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.875 and backtrack 1
.Update replanning_limit_time 600.99, with current_time : 591.953 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.967 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 591.98 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.015 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.027 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.037 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.048 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.072 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.118 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.143 and backtrack 1
.Update replanning_limit_time 600.99, with current_time : 592.239 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.272 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.297 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.32 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.34 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.368 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.38 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.392 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.42 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.436 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.446 and backtrack 1
.Update replanning_limit_time 600.99, with current_time : 592.542 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.573 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.588 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.601 and backtrack 1
.Update replanning_limit_time 600.99, with current_time : 592.773 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.811 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.832 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.848 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.863 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.902 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 592.921 and backtrack 1
.Update replanning_limit_time 600.99, with current_time : 593.025 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.073 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.101 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.117 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.156 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.18 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.193 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.203 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.213 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.263 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.293 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.306 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.319 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.33 and backtrack 0
Update replanning_limit_time 600.99, with current_time : 593.344 and backtrack 0
nb of planning steps: 64
SOLUTION FOUND
REPLANNING Total Duration : 2.196
Start cycle at 593.489
Update ExecTPs : [3]
########### ACT #############
Instantiation of EndPlanning timepoint 3 at 593.489
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [4] at 600
CYCLE DURATION : 0.048

CYCLE N°14 - WAKE UP AT : 600
########### ACT #############
Instantiation of Extern timepoint 4 at 600
Update ExecTPs : [17]
Instantiation of Start timepoint 17 at 600.01
Send-command : (LAUNCH IXTET_EXEC COMMUNICATE 23 (PARAMETERS W2))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [18] at 719.99
CYCLE DURATION : 0.169

MsgQueue not empty : [MsgBilan (BILAN, Task 23, nominal, state bilan[] , resource bilan  [(CHANNEL() 1)]), timepoint 18]
```

```
CYCLE N°15 - WAKE UP AT : 704.206
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 700.02
Timepoint upper_bound : 721.99
Instantiation of Msg timepoint 18 at 704.206
Update ExecTPs : [21, 11]
########### ACT #############
Instantiation of Start timepoint 21 at 704.216
Send-command : (LAUNCH IXTET_EXEC MOVE 25 (PARAMETERS 10.182649 -1.707548 8 -5))
Update ExecTPs : [11]
Instantiation of Goal timepoint 11 at 704.216
Update ExecTPs : [12]
Instantiation of Goal timepoint 12 at 704.226
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [5] at 720
CYCLE DURATION : 0.105

CYCLE N°16 - WAKE UP AT : 720
########### ACT #############
Instantiation of Extern timepoint 5 at 720
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [22] at 884.719
CYCLE DURATION : 0.013

CYCLE N°17 - WAKE UP AT : 884.72
########### ACT #############
Send-command : (END 25)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.034

MsgQueue not empty : [MsgBilan (BILAN, Task 25, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -0.727463), (AT_ROBOT_X() 9.91046), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 22]

CYCLE N°18 - WAKE UP AT : 888.946
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 742.542
Timepoint upper_bound : 886.719
New timepoint 33
Waiting End Timepoint 33 TIMED OUT
New state insertion...
19 causal links are removed on the attributes:
[MVT_GENERATION_INITIALIZED(), AT_ROBOT_X(), AT_ROBOT_Y(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION()]
Update ExecTPs : [25]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 33
... Solution found
Update ExecTPs : []
########### ACT #############
WaitingExecTPs : []
############################
Next ExecTPs [34] at 890.956
CYCLE DURATION : 1.153

CYCLE N°19 - WAKE UP AT : 890.956
########### ACT #############
Instantiation of Start timepoint 34 at 890.956
Send-command : (LAUNCH IXTET_EXEC MOVE 31 (PARAMETERS 9.910457 -0.727463 8 -5))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [35] at 1095.06
CYCLE DURATION : 0.047

MsgQueue not empty : [MsgBilan (BILAN, Task 31, nominal, state bilan[] , resource bilan  [(CAMERA() 1)]), timepoint 35]

CYCLE N°20 - WAKE UP AT : 1060.02
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 934.237
Timepoint upper_bound : 1097.06
Instantiation of Msg timepoint 35 at 1060.02
Update ExecTPs : [25]
```

```
########### ACT #############
Instantiation of Start timepoint 25 at 1060.03
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 27 (PARAMETERS FORWARD DOWNWARD))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [26] at 1068.03
CYCLE DURATION : 0.058

MsgQueue not empty : [MsgBilan (BILAN, Task 27, nominal, state bilan[] , resource bilan  []), timepoint 26]

CYCLE N°21 - WAKE UP AT : 1064.48
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1063.03
Timepoint upper_bound : 1070.03
Instantiation of Msg timepoint 26 at 1064.48
Update ExecTPs : [13]
########### ACT #############
Instantiation of Start timepoint 13 at 1064.49
Send-command : (LAUNCH IXTET_EXEC TAKE_PICTURE 21 (PARAMETERS OBJ3 8 -5))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [14] at 1124.49
CYCLE DURATION : 0.062

MsgQueue not empty : [MsgBilan (BILAN, Task 21, nominal, state bilan[] , resource bilan  [(CAMERA() 1),
(STORAGE_CPT() 41699), (STORAGE() 24301)]), timepoint 14]

CYCLE N°22 - WAKE UP AT : 1066.48
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1066.5
Timepoint upper_bound : 1124.49
New timepoint 36
Failed timepoints : [14]
New state insertion...
26 causal links are removed on the attributes:
[AT_ROBOT_X(), MVT_GENERATION_INITIALIZED(), AT_ROBOT_Y(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION(), PICTURE(OBJ3,8 ,-5 )]
Update ExecTPs : [29, 7]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 27
... Solution found
Update ExecTPs : [29, 7]
########### ACT #############
Instantiation of Start timepoint 29 at 1066.49
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 29 (PARAMETERS DOWNWARD FORWARD))
Update ExecTPs : [7]
Instantiation of Goal timepoint 7 at 1066.49
Update ExecTPs : [8]
Instantiation of Goal timepoint 8 at 1066.5
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [30] at 1074.49
CYCLE DURATION : 1.106

MsgQueue not empty : [MsgBilan (BILAN, Task 29, nominal, state bilan[] , resource bilan  []), timepoint 30]

CYCLE N°23 - WAKE UP AT : 1071.01
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1069.49
Timepoint upper_bound : 1076.49
Instantiation of Msg timepoint 30 at 1071.01
Update ExecTPs : [19]
########### ACT #############
Instantiation of Start timepoint 19 at 1071.02
Send-command : (LAUNCH IXTET_EXEC MOVE 24 (PARAMETERS 8 -5 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [20] at 1252.35
CYCLE DURATION : 0.065

CYCLE N°24 - WAKE UP AT : 1252.35
########### ACT #############
Send-command : (END 24)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
```

```
CYCLE DURATION : 0.023

MsgQueue not empty : [MsgBilan (BILAN, Task 24, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -3.86292), (AT_ROBOT_X() 8.09759), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 20]

CYCLE N°25 - WAKE UP AT : 1256.48
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1109.52
Timepoint upper_bound : 1254.35
New timepoint 37
Waiting End Timepoint 37 TIMED OUT
New state insertion...
9 causal links are removed on the attributes:
[MVT_GENERATION_INITIALIZED(), AT_ROBOT_Y(), AT_ROBOT_X(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION()]
Update ExecTPs : [27]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 23
... Solution found
Update ExecTPs : []
########### ACT #############
WaitingExecTPs : []
############################
Next ExecTPs [38] at 1258.49
CYCLE DURATION : 0.965

CYCLE N°26 - WAKE UP AT : 1258.49
########### ACT #############
Instantiation of Start timepoint 38 at 1258.49
Send-command : (LAUNCH IXTET_EXEC MOVE 32 (PARAMETERS 8.097589 -3.862917 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [39] at 1398.67
CYCLE DURATION : 0.088

CYCLE N°27 - WAKE UP AT : 1398.68
########### ACT #############
Send-command : (END 32)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.05

MsgQueue not empty : [MsgBilan (BILAN, Task 32, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -2.13912), (AT_ROBOT_X() 5.73907), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 39]

CYCLE N°28 - WAKE UP AT : 1403.01
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1288.35
Timepoint upper_bound : 1400.67
New timepoint 40
Waiting End Timepoint 40 TIMED OUT
New state insertion...
9 causal links are removed on the attributes:
[MVT_GENERATION_INITIALIZED(), AT_ROBOT_Y(), AT_ROBOT_X(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION()]
Update ExecTPs : [27]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 23
... Solution found
Update ExecTPs : []
########### ACT #############
WaitingExecTPs : []
############################
Next ExecTPs [41] at 1405.02
CYCLE DURATION : 1.213

CYCLE N°29 - WAKE UP AT : 1405.02
########### ACT #############
Instantiation of Start timepoint 41 at 1405.02
Send-command : (LAUNCH IXTET_EXEC MOVE 33 (PARAMETERS 5.739069 -2.139120 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [42] at 1566.35
CYCLE DURATION : 0.055
CYCLE N°30 - WAKE UP AT : 1566.35
########### ACT #############
Send-command : (END 33)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.025

MsgQueue not empty : [MsgBilan (BILAN, Task 33, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() -2.13628), (AT_ROBOT_X() 5.52935), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 42]

CYCLE N°31 - WAKE UP AT : 1571.08
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1439.32
Timepoint upper_bound : 1568.35
New timepoint 43
Waiting End Timepoint 43 TIMED OUT
New state insertion...
9 causal links removed on the attributes:
[MVT_GENERATION_INITIALIZED(), AT_ROBOT_Y(), AT_ROBOT_X(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION()]
Update ExecTPs : [27]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 17
... No solution found
Update ExecTPs : []
########### ACT #############
WaitingExecTPs : []
############################
Next ExecTPs [44] at 1573.09
CYCLE DURATION : 1.245

CYCLE N°32 - WAKE UP AT : 1572.32
########### PLAN REPAIR #############
nb planning steps: 6
... Solution found
Update ExecTPs : [44]
########### ACT #############
Instantiation of Start timepoint 44 at 1573.09
Send-command : (LAUNCH IXTET_EXEC MOVE 34 (PARAMETERS 5.529349 -2.136283 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [45] at 1741.32
CYCLE DURATION : 0.361

MsgQueue not empty : [MsgBilan (BILAN, Task 34, nominal, state bilan[] , resource bilan  [(CAMERA() 1)]), timepoint 45]

CYCLE N°33 - WAKE UP AT : 1718.42
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1608.83
Timepoint upper_bound : 1743.32
Instantiation of Msg timepoint 45 at 1718.42
Update ExecTPs : [27]
########### ACT #############
Instantiation of Start timepoint 27 at 1718.43
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 28 (PARAMETERS FORWARD DOWNWARD))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [28] at 1726.43
CYCLE DURATION : 0.092

MsgQueue not empty : [MsgBilan (BILAN, Task 28, nominal, state bilan[] , resource bilan  []), timepoint 28]

CYCLE N°34 - WAKE UP AT : 1722.88
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1721.43
Timepoint upper_bound : 1728.43
Instantiation of Msg timepoint 28 at 1722.88
Update ExecTPs : [15]
########### ACT #############
Instantiation of Start timepoint 15 at 1722.89
Send-command : (LAUNCH IXTET_EXEC TAKE_PICTURE 22 (PARAMETERS OBJ2 9 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [16] at 1782.89
CYCLE DURATION : 0.068
```

```
MsgQueue not empty : [MsgBilan (BILAN, Task 22, nominal, state bilan[] , resource bilan  [(CAMERA() 1),
(STORAGE_CPT() 63948), (STORAGE() 2052)]), timepoint 16]

CYCLE N°35 - WAKE UP AT : 1724.73
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1724.9
Timepoint upper_bound : 1782.89
New timepoint 46
Failed timepoints : [16]
New state insertion...
16 causal links are removed on the attributes:
[AT_ROBOT_X(), MVT_GENERATION_INITIALIZED(), AT_ROBOT_Y(), ROBOT_STATUS(), PTU_DRIVER_INITIALIZED(),
PAN_TILT_UNIT_POSITION(), PICTURE(OBJ2,9,0)]
Update ExecTPs : [31, 9]
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 13
... No solution found
Update ExecTPs : [31, 9]
########### ACT #############
Timepoint 31 is not supported, upper_bound : 1935.48
-> WaitingExecTPs
Update ExecTPs : [31, 9]
Instantiation of Goal timepoint 9 at 1724.74
Update ExecTPs : [10]
Instantiation of Goal timepoint 10 at 1724.75
Update ExecTPs : []
WaitingExecTPs : [31]
############################
Next ExecTPs [31] at 1724.74
CYCLE DURATION : 1.335

CYCLE N°36 - WAKE UP AT : 1726.06
########### PLAN REPAIR #############
New Repair Search Tree
nb planning steps: 2
... Solution found
Update ExecTPs : [31]
########### ACT #############
Instantiation of Start timepoint 31 at 1726.06
Send-command : (LAUNCH IXTET_EXEC MOVE_PAN_TILT_UNIT 30 (PARAMETERS DOWNWARD FORWARD))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [32] at 1734.06
CYCLE DURATION : 0.292

MsgQueue not empty : [MsgBilan (BILAN, Task 30, nominal, state bilan[] , resource bilan  []), timepoint 32]

CYCLE N°37 - WAKE UP AT : 1730.9
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1729.06
Timepoint upper_bound : 1736.06
Instantiation of Msg timepoint 32 at 1730.9
Update ExecTPs : [23]
########### ACT #############
Instantiation of Start timepoint 23 at 1730.91
Send-command : (LAUNCH IXTET_EXEC MOVEX 26 (PARAMETERS 9 0 0 0))
Update ExecTPs : []
WaitingExecTPs : []
############################
Next ExecTPs [24] at 1995.99
CYCLE DURATION : 0.074

CYCLE N°38 - WAKE UP AT : 1996.01
########### ACT #############
Send-command : (END 26)
WaitingExecTPs : []
############################
Next ExecTPs [] at 1.79769e+308
CYCLE DURATION : 0.072

MsgQueue not empty : [MsgBilan (BILAN, Task 26, interrupted, state bilan[(PAN_TILT_UNIT_POSITION() FORWARD),
(MVT_GENERATION_INITIALIZED() T), (AT_ROBOT_Y() 0.852485), (AT_ROBOT_X() 4.66933), (ROBOT_STATUS() STILL)] ,
resource bilan  [(CAMERA() 1)]), timepoint 24]

CYCLE N°39 - WAKE UP AT : 2000.68
########### SENSE #############
# Receive bilan
Timepoint lower_bound : 1790.41
Timepoint upper_bound : 1997.99
```

```
New timepoint 47
Waiting End Timepoint 47 TIMED OUT
NO TEMPORAL FLEXIBILITY LEFT, replan
REPLANNING ATTEMPT N°1
with 0 goals.
Could not find a new plan - NO MORE GOAL
```

Bibliography

[Ai-Chang 03] M. Ai-Chang, J. Bresina, L. Charest, A. Jónsson, J. Hsu, B. Kanefsky, P. Maldague, P. Morris, K. Rajan & J. Yglesias. *MAPGEN: Mixed initiative planning and scheduling for the Mars 03 MER mission.* In International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS), 2003.

[Alami 98] R. Alami, R. Chatila, S. Fleury, M. Ghallab & F. Ingrand. *An Architecture for Autonomy.* International Journal of Robotics Research, Special Issue on Integrated Architectures for Robot Control and Programming, vol. 17, no. 4, 1998.

[Allen 84] J. Allen. *Towards a General Theory of Action and Time.* Artificial Intelligence, vol. 23, 1984.

[Ambros-Ingerson 88] J. Ambros-Ingerson & S. Steel. *Integrating Planning, Execution and Monitoring.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1988.

[Bacchus 01] F. Bacchus & M. Ady. *Planning with Resources and Concurrency: A Forward Chaining Approach.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Bastié 96] C. Bastié & P. Régnier. *Intégration de la planification et du contrôle d'exécution en environnement dynamique : le système SPEEDY.* In Actes de RFIA, Reconnaissance des Formes et Intelligence Artificielle, 1996.

[Beetz 94] M. Beetz & D. V. McDermott. *Improving Robot Plans During Their Execution.* In Proceedings of the International Conference on AI Planning Systems (AIPS), 1994.

[Bertoli 01] P. Bertoli, A. Cimatti, M. Roveri & P. Traverso. *Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Bonasso 97] R. P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. P. Miller & M. G. Slack. *Experiences with an Architecture for Intelligent, Reactive Agents.* Journal of Experimental & Theoretical Artificial Intelligence, vol. 9, no. 2/3, pages 237–256, April 1997.

[Bonet 00] B. Bonet & H. Geffner. *Planning with Incomplete Information as Heuristic Search in Belief Space.* In Proceedings of the International Conference on AI Planning Systems (AIPS), 2000.

[Cambon 02] S. Cambon, S. Lemai & R. Trinquart. *Compétitions de planification.* Revue d'Intelligence Artificielle, volume 16, n°3, 2002.

[Chien 00] S. Chien, R. Knight, A. Stechert, R. Sherwood & G. Rabideau. *Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling.* In Proceedings of the International Conference on AI Planning Systems (AIPS), 2000.

[Cimatti 00] A. Cimatti & M. Roveri. *Conformant Planning via Symbolic Model Checking.* Journal of Artificial Intelligence Research, vol. 13, pages 305–338, 2000.

[Coddington 02] A. Coddington. *A Continuous Planning Framework with Durative Actions.* Ninth International Symposium on Temporal Representation and Reasoning (TIME'02), 2002.

[Currie 91] K. Currie & A. Tate. *O-Plan: the Open Planning Architecture.* Artificial Intelligence, vol. 52, 1991.

[Dechter 89] Rina Dechter, Itay Meiri & Judea Pearl. *Temporal Constraint Networks.* In KR'89: Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, 1989.

[Despouys 99] O. Despouys & F. Ingrand. *Propice-Plan: Toward a Unified Framework for Planning and Execution.* In Proceedings of the European Conference on Planning (ECP), 1999.

[Dias 03] M. B. Dias, S. Lemai & N. Muscettola. *A Real-Time Rover Executive Based On Model-Based Reactive Planning.* In Proceedings of the International Conference on Advanced Robotics (ICAR), 2003.

[Do 01] M. Do & S. Kambhampati. *Sapa: A Domain-Independent Heuristic Metric Temporal Planner.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Do 03] M. Do & S. Kambhampati. *Improving the Temporal Flexibility of Position Constrained Metric Temporal Plans.* In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2003.

[Doherty 01] P. Doherty & J. Kvarnström. *TALplanner: A Temporal Logic Based Planner.* AI magazine, vol. 22, no. 3, 2001.

[Edelkamp 01] S. Edelkamp. *First Solutions to PDDL+ Planning Problems.* In PlanSig Workshop, 2001.

[Estlin 01] T. Estlin, R. Volpe, I. Nesnas, D. Mutz, F. Fisher, B. Engelhardt & S. Chien. *Decision-Making in a Robotic Architecture for Autonomy.*

In International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS), 2001.

[Fargier 96] H. Fargier, J. Lang & T. Schiex. *Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.

[Ferraris 00] P. Ferraris & E. Giunchiglia. *Planning as Satisfiability in Nondeterministic Domains.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2000.

[Fikes 71] R.E. Fikes & N.J. Nilsson. *STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving.* Artificial Intelligence, vol. 2, 1971.

[Fikes 72] R.E. Fikes, P. E. Hart & N. J.Nilsson. *Learning and executing generalized robot plans.* Artificial Intelligence, vol. 3, 1972.

[Finzi 04] A. Finzi, F. Ingrand & N. Muscettola. *Robot Action Planning and Execution Control.* In International Workshop on Planning and Scheduling for Space, 2004.

[Firby 94] R. J. Firby. *Task Networks for Controlling Continuous Processes.* In Artificial Intelligence Planning Systems, 1994.

[Fleury 94] S. Fleury, M. Herrb & R. Chatila. *Design of a Modular Architecture for Autonomous Robot.* In Proceedings of the International Conference on Robotics and Automation (ICRA), 1994.

[Fox 02a] M. Fox & D. Long. *International Planning Competition.* http://www.dur.ac.uk/d.p.long/competition.html, 2002.

[Fox 02b] Maria Fox & Derek Long. *PDDL 2.1 : An Extension to PDDL for Expressing Temporal Planning Domains.* Technical Report, University of Durham, UK, 2002.

[Frank 03] J. Frank & A. Jónsson. *Constraint-Based Attribute and Interval Planning.* Constraints, vol. 8, no. 4, 2003.

[Gaborit 96] P. Gaborit. *Planification distribuée pour la coopération multiagents.* PhD Thesis, Université Paul Sabatier de Toulouse, 1996.

[Gallien 04] M. Gallien. *Planification et contrôle d'exécution.* Master Thesis, Université Paul Sabatier de Toulouse, 2004.

[Garcia 95] F. Garcia & P. Laborie. *Hierarchisation of the search space in temporal planning.* In Proceedings of the European Workshop on Planning (EWSP), 1995.

[Garcia 98] F. Garcia & P. Laborie. *Hiérarchisation dynamique de la recherche: Application au planificateur IxTeT.* Revue d'Intelligence Artificielle, vol. 12, 1998.

[Gat 92] E. Gat. *Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1992.

[Gat 97] E. Gat. *ESL : A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents.* In Proceedings of the 1997 IEEE Aerospace Conference, 1997.

[Georgeff 89] M. P. Georgeff & F. Ingrand. *Decision Making in an Embedded Reasoning System.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1989.

[Ghallab 89] M. Ghallab & A. Mounir-Alaoui. *Managing Efficiently Temporal Relations Through Indexed Spanning Trees.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1989.

[Ghallab 94] M. Ghallab & H. Laruelle. *Representation and Control in IxTeT, a Temporal Planner.* In Proceedings of the International Conference on AI Planning Systems (AIPS), 1994.

[Ghallab 01] M. Ghallab, F. Ingrand, S. Lemai & F. Py. *Architecture and Tools for Autonomy in Space.* In International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS), 2001.

[Ghallab 04] M. Ghallab, D. Nau & P. Traverso. Automated planning: Theory and practice. Morgan Kaufmann, 2004.

[Gout 99] J. Gout, S. Fleury & H. Schindler. *A New Design Approach of Software Architecture for an Autonomous Observation Satellite.* In International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS), 1999.

[Grandjean 04] P. Grandjean & A. Pitie. *Operational multi-mission spacecraft operations on-line scheduling and automated execution.* In International Workshop on Planning and Scheduling for Space, 2004.

[Guéré 01] E. Guéré. *Extraire la structure des domaines pour planifier en présence d'incertitudes.* PhD Thesis, Université Paul Sabatier de Toulouse, 2001.

[Haigh 98] K. Z. Haigh & M. M. Veloso. *Planning, Execution and Learning in a Robotic Agent.* In Proceedings of the International Conference on AI Planning Systems (AIPS), 1998.

[Huguet 02] M.-J. Huguet, P. Lopez & T. Vidal. *Dynamic task sequencing in temporal problems with uncertainty.* In WS On-line Planning and Scheduling, AIPS, 2002.

[Infantes 03] G. Infantes. *Apprentissage Actif et Planification en Robotique.* Master Thesis, Université Paul Sabatier de Toulouse, 2003.

[Ingrand 96] F. Ingrand, R. Chatila, R. Alami & F. Robert. *PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots.* In Proceedings of the International Conference on Robotics and Automation (ICRA), 1996.

[Ingrand 02] F. Ingrand & F. Py. *An Execution Control System for Autonomous Robots.* In Proceedings of the International Conference on Robotics and Automation (ICRA), 2002.

[Kim 01] P. Kim, B. Williams & M. Abramson. *Executing Reactive, Model-based Programs through Graph-based Temporal Planning.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Knoblock 94] C. Knoblock. *Automatically generating abstractions for planning.* Artificial Intelligence, vol. 68, 1994.

[Kortenkamp 98] D. Kortenkamp, R. Bonasso & R. Murphy. Artificial intelligence and mobile robots: Case studies of successful robot systems. AAAI Press, 1998.

[Laborie 95a] P. Laborie. *IxTeT : une approche intégrée pour la gestion de ressources et la synthèse de plans.* PhD Thesis, Ecole Nationale Supérieure des Télécommunications, 1995.

[Laborie 95b] P. Laborie & M. Ghallab. *Planning with Sharable Resource Constraints.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1995.

[Laborie 01] P. Laborie. *Algorithms for Propagating Resource Constraints in A.I. Planning and Scheduling: Existing Approaches and New Results.* Proceedings of the European Conference on Planning (ECP), 2001.

[Lacroix 02] S. Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb & R. Chatila. *Autonomous Rover Navigation on Unknown Terrains: Functions and Integration.* International Journal of Robotics Research, vol. 21, no. 10-11, 2002.

[Laruelle 94] H. Laruelle. *Planification temporelle et exécution de tâches en robotique.* PhD Thesis, Université Paul Sabatier de Toulouse, 1994.

[Lemai 02] S. Lemai & R. Trinquart. *Bringing out IxTeT in the dynamic world?* In WS On-line Planning and Scheduling, AIPS, 2002.

[Lemai 03] S. Lemai & F. Ingrand. *Interleaving Temporal Planning and Execution: IXTET-EXEC.* In WS Plan Execution, ICAPS, 2003.

[Lemai 04a] S. Lemai & F. Ingrand. *Interleaving Temporal Planning and Execution for an Autonomous Rover.* In International Workshop on Planning and Scheduling for Space, 2004.

[Lemai 04b] S. Lemai & F. Ingrand. *Interleaving Temporal Planning and Execution in Robotics Domains.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2004.

[Lemai 04c] S. Lemai & F. Ingrand. *Planification et contrôle d'exécution temporels : IXTET-EXEC.* In Actes de RFIA, Reconnaissance des Formes et Intelligence Artificielle, 2004.

[Levinson 95] R. Levinson. *A General Programming Language for Unified Planning and Control.* Artificial Intelligence, vol. 76, 1995.

[Morisset 02] B. Morisset & M. Ghallab. *Learning How to Combine Sensory-Motor Modalities for a Robust Behavior.* Advances in Plan-Based Control of Robotic Agents. M. Beetz, J. Hertzberg, M. Ghallab, M. E. Pollack - Springer, vol. 2466, 2002.

[Morris 01] P. Morris, N. Muscettola & T. Vidal. *Dynamic Control of Plans with Temporal Uncertainty.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Muscettola 94] N. Muscettola. *HSTS: Integrated planning and scheduling.* In Fox, M., and Zweben, M., eds, Intelligent Scheduling, Morgan Kaufman, 1994.

[Muscettola 98] N. Muscettola, P. P. Nayak, B. Pell & B. Williams. *Remote Agent : To Boldly Go Where No AI System Has Gone Before.* Artificial Intelligence, vol. 103, 1998.

[Muscettola 02] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson & C. Plaunt. *IDEA: Planning at the Core of Autonomous Reactive Agents.* In International NASA Workshop on Planning and Scheduling for Space, 2002.

[Myers 99] Karen L. Myers. *CPEF: Continuous Planning and Execution Framework.* AI Magazine, vol. 20, no. 4, 1999.

[Nau 01] D. Nau, H. Mun oz Avila, Y. Cao, A. Lotem & S. Mitchell. *Total-Order Planning with Partially Ordered Subtasks.* Proceedings of the

International Joint Conference on Artificial Intelligence (IJCAI), 2001.

[Pell 97] B. Pell, E. Gat, R. Keesing, N. Muscettola & B. Smith. *Robust Periodic Planning and Execution for Autonomous Spacecraft.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1997.

[Penberthy 94] J. S. Penberthy & D. Weld. *Planning with Continuous Change.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1994.

[Rabideau 99] G. Rabideau, R. Knight, S. Chien, A. Fukunaga & A. Govindjee. *Iterative Repair Planning for Spacecraft Operations in the ASPEN System.* In International Symposium on Artificial Intelligence Robotics and Automation in Space (iSAIRAS), 1999.

[Sacerdoti 74] E. Sacerdoti. *Planning in a Hierarchy of Abstraction Spaces.* Artificial Intelligence, vol. 5, 1974.

[Schwalb 97] E. Schwalb & R. Dechter. *Processing Disjunctions in Temporal Constraint Networks.* Artificial Intelligence, vol. 93, pages 29–61, 1997.

[Shapiro 99] R. Shapiro, Y. Feldman & R. Dechter. *On the Complexity of Interval-Based Constraint Networks.* In MISC'99 Workshop on Applications of Interval Analysis to Systems and Control, 1999.

[Simmons 98] R. Simmons & D. Apfelbaum. *A Task Description Language for Robot Control.* In Proceedings of Conference on Intelligent Robotics and Systems, 1998.

[Smith 98] D. Smith & D. Weld. *Conformant Graphplan.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998.

[Smith 99] D. Smith & D. Weld. *Temporal Planning with Mutual Exclusion Reasoning.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[Smith 00] D. Smith, J. Frank & A. Jónsson. *Bridging the Gap Between Planning and Scheduling.* Knowledge Engineering Review, 15(1):61–94, 2000.

[Stone 96] P. Stone & M. M. Veloso. *User-guided Interleaving of Planning and Execution.* In New Directions in AI Planning. IOS Press, 1996.

[Trinquart 01] Romain Trinquart & Malik Ghallab. *An Extended Functional Representation in Temporal Planning : towards Continuous Change.* In Proceedings of the European Conference on Planning (ECP), 2001.

[Trinquart 02] R. Trinquart, S. Lemai & S. Cambon. *One step on the left, one step on the right and back to the middle: exploring temporal domains in a POP fashion.* In WS Planning for Temporal Domains, AIPS, 2002.

[Trinquart 04] R. Trinquart. *Planification en Robotique: Analyse d'accessibilité dans l'espace des plans partiels.* PhD Thesis, Institut National Polytechnique de Toulouse, 2004.

[Tsamardinos 98] I. Tsamardinos, N. Muscettola & P. Morris. *Fast Transformation of Temporal Plans for Efficient Execution.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998.

[Vere 83] S. Vere. *Planning in Time: Windows and Durations for Activities and Goals.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 5, 1983.

[Vidal 99] T. Vidal & H. Fargier. *Handling contingency in temporal constraint networks: from consistency to controllabilities.* Journal of Experimental and and Theoretical Artificial Intelligence, vol. 11, 1999.

[Vilain 86] M. Vilain & H. Kautz. *Constraint Propagation Algorithms for Temporal Reasoning.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1986.

[Volpe 00] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras & H. Das. *CLARAty : Coupled Layer Architecture for Robotic Autonomy.* Technical Report, Jet Propulsion Laboratory, Dec. 2000.

[Vossen 99] T. Vossen, M. Ball, A. Lotem & D. Nau. *On the Use of Integer Programming Models in AI Planning.* In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1999.

[Washington 99] R. Washington, K. Golden & J. Bresina. *Plan Execution, Monitoring and Adaptation for Planetary Rovers.* In Proceedings of the IJCAI-99 Workshop, Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World., 1999.

[Weld 98] D. Weld, C. Anderson & D. Smith. *Extending Graphplan to Handle Uncertainty and Sensing Actions.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1998.

[Wilkins 88] D. E. Wilkins. Practical planning: Extending the classical ai planning paradigm. Morgan Kaufmann, 1988.

[Williams 96] B. Williams & P. Nayak. *A Model-based Approach to Reactive Self-Configuring Systems.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1996.

[Yang 90] Q. Yang & J. D. Tenenberg. *ABTWEAK: Abstracting a nonlinear, least commitment planner.* In Proceedings of the National Conference on Artificial Intelligence (AAAI), 1990.