
Automated Hierarchical, Forward-Chaining Temporal Planner for Planetary Robots Exploring Unknown Environments



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vom Fachbereich Informatik der
Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte

Dissertation

von

M.Sc.-Inform. Juan Manuel Delfa Victoria
(geboren in Viso del Marqués, Spanien)

Referent: Prof. Dr. Oskar von Stryk
Koreferentin: Prof. Dr. Yang Gao
(University of Surrey, England)
Koreferent: Dr. Nicola Policella
(European Space Agency, Germany)

Tag der Einreichung: 19.05.2015
Tag der mündlichen Prüfung: 17.06.2015

D17
Darmstadt 2016

Please cite this document as

URN: urn:nbn:de:tuda-tuprints-FÜNFSTELLIGE NUMMER - VIERSTELLIGE EINTRAGSNUMMER PLUS PRÜFZIFFER

URL: <http://tuprints.ulb.tu-darmstadt.de/VIERSTELLIGE EINTRAGSNUMMER>

This document is provided by tuprints,
E-Publishing-Service of the TU Darmstadt
<http://tuprints.ulb.tu-darmstadt.de>
tuprints@ulb.tu-darmstadt.de

Contents

1. Introduction	5
1.1. Automated Planning: from Theory to Practice	7
1.2. Motivation and Contributions	9
1.3. Outline of the Thesis	10
2. Automated Planning and Execution: Background	13
2.1. Classical vs. Applied Planning	13
2.2. Planning Domains	15
2.2.1. Classical Domains	15
2.2.2. Applied Domains	16
2.3. Planning Formalisms	26
2.3.1. State-Space	27
2.3.2. Hierarchical Task Networks (HTN)	31
2.3.3. Timeline Planning	33
2.3.4. Planning under Uncertainty	36
2.3.5. Constraint Satisfaction Problems (CSP)	38
2.4. Software for Autonomy	39
2.4.1. Search Algorithms for Planning	39
2.4.2. Forward-Chaining Planners	44
2.4.3. HTN Planners	45
2.4.4. Timeline Frameworks and Planners	46
2.4.5. Expert Systems for Rovers	52
2.5. Plan Execution	54
2.5.1. T-REX	56
2.5.2. IDEA - Plan Runner	57
2.5.3. Universal Executive (UE)	57
2.5.4. Teer	57
2.5.5. SMACH	58
2.6. Benchmarking Methodology	58
2.6.1. Planning Performance	59
2.6.2. Expressiveness	59
2.6.3. Robustness	59
2.6.4. Execution Power	59
2.7. Conclusions	60
3. Hierarchical Timeline Networks: A Novel Planning Formalism	63
3.1. Background	63
3.1.1. Combining HTN and Temporal Planning Theories	63
3.1.2. Hypergraph Theory	64
3.2. HTLN Formalism	66
3.2.1. Values and Decisions	66

3.2.2.	Decision Network (dn)	67
3.2.3.	Primitive Value (v^p)	67
3.2.4.	Complex Value (v^c)	68
3.2.5.	Domain (D)	69
3.2.6.	Problem (P)	69
3.2.7.	Method (m)	70
3.2.8.	Relations (f)	72
3.2.9.	Resolver (ρ)	72
3.2.10.	Transition Function (γ)	72
3.2.11.	Unification	73
3.2.12.	Decomposition Resolver (ρ_δ)	76
3.3.	Properties of HTLN	79
3.3.1.	Soundness and Completeness	80
3.3.2.	Robustness	81
3.3.3.	Performance	83
3.3.4.	Expressiveness	88
3.4.	Conclusions	89
4.	QuijoteExpress Planner: A Novel Planning System	91
4.1.	Architecture	91
4.2.	APSI*	92
4.2.1.	Expanding the APSI Core	93
4.2.2.	Search Space	94
4.2.3.	Solvers	96
4.3.	QuijoteExpress	98
4.3.1.	Configuration & Preprocessor	100
4.3.2.	Supersolver	102
4.3.3.	Resolvers	106
4.3.4.	Heuristics	111
4.3.5.	Search Enhancements	113
4.4.	Evaluation	115
4.4.1.	Testing QEv1	115
4.4.2.	Testing QEv2	118
4.5.	Conclusions	121
5.	SanchoExpress: Flexible Execution with FDIR Capabilities	123
5.1.	Architecture	123
5.2.	Components	124
5.2.1.	User	125
5.2.2.	RobCon	126
5.2.3.	Planner	127
5.2.4.	SanchoExpress	127
5.2.5.	Dedicated Executive	131
5.3.	Conclusions	132

6. Evaluation in Real-World Scenarios	135
6.1. Tailoring the Planner and Executive	135
6.1.1. Adapting QuijoteExpress	135
6.1.2. Adapting SanchoExpress	136
6.2. Impact of QuijoteExpress and SanchoExpress Features on the Modelling	137
6.3. The FASTER Project	138
6.3.1. Introduction to Rover Missions	138
6.3.2. Description of the Mission	138
6.3.3. Modelling the Planner Inputs for FASTER	138
6.3.4. Implementing the Dedicated Executives for FASTER	150
6.3.5. Evaluation	150
6.4. The USAR Project	156
6.4.1. Description of the Mission	157
6.4.2. Modelling the Planner Inputs for USAR	157
6.4.3. Dedicated Executives for USAR	160
6.4.4. Evaluation	160
6.5. Conclusions	162
7. Conclusions	165
7.1. Summary	165
7.1.1. HTLN	165
7.1.2. QuijoteExpress	165
7.1.3. SanchoExpress	166
7.2. Future Work	166
7.2.1. Future of Planning for Robotic Applications	166
7.2.2. Future of Execution on New Robotic Applications	168
7.3. Closing Remarks	168
A. Appendices	171
A.1. HTLN nomenclature	172
Bibliography	173
Own Publications	183



List of Figures

1.1. Mars rover exploration problem.	8
2.1. Sojourner, MER and Curiosity Mars rovers (Courtesy of JPL).	18
2.2. Mars Sample Return mission (Courtesy of NASA).	22
2.3. Search space covering	30
2.4. SMA*	42
2.5. Primary Locomotion State Variable.	51
2.6. MER Uplink process (Courtesy of J. Richard Morris and NASA-JPL).	55
3.1. Hypergrah (Courtesy of Wikipedia).	65
3.2. Decomposition methods for <i>Traverse</i> complex goal in FASTER scenario.	84
4.1. QuijoteExpress packages.	92
4.2. Search node states.	95
4.3. Two-levels bucket-based priority queue (Courtesy of IBM).	96
4.4. Open-list structure.	97
4.5. APSI* algorithms class diagram.	98
4.6. Next resolver strategy - UDS (Unfolder, Decomposer, Scheduler).	103
4.7. Next resolver strategy - USDS (Unfolder, Scheduler, Decomposer, Scheduler).	103
4.8. Propagation of a <i>DEAD_END</i> subtask up to the supertasks.	105
4.9. Problem division in subproblems.	105
4.10. Example of a matching frontier.	108
4.11. Selection of timelines for which the unfolder needs to search successors.	108
4.12. Component A* Search.	109
4.13. Hierarchical rover model.	115
4.14. Knowledge database used to evaluate QEv1.	116
4.15. Performance of QEv1 and AP2 in $RD - C$	117
4.16. Performance of QEv1 and AP2 in $RD - S$	117
4.17. Number of nodes generated by QEv1 and AP2 in $RD - C$	118
4.18. Number of nodes generated by QEv1 and AP2 in $RD - S$	118
4.19. Performance of QEv2 and AP ² in Totally Ordered Problems.	119
4.20. Performance of QEv2 and AP ² in Partially Ordered Problems.	119
4.21. CPU usage by QEv2 resolvers.	120
5.1. Three layers architecture with QE and SE.	123
5.2. Multi-rover architecture with replanning capabilities. Each box is a ROS node: Blue boxes represent deliberative components, green reactive and grey functional. The orange boxes on top of the arrows indicate service calls.	125
5.3. Life cycle of a plan.	126
5.4. Example of how SanchoExpress executes a locomotion activity.	131
6.1. Bridget rover (Courtesy of Astrium UK).	139
6.2. Scout Rover (Courtesy of DFKI).	139

6.3. Faster subsystems.	140
6.4. Faster Behaviours.	144
6.5. Primary and Scout Rovers in formation (Courtesy of Airbus).	145
6.6. Path Planning for MSL (Courtesy of SAS).	146
6.7. Merged point clouds from primary and scout rovers (Courtesy of SAS).	148
6.8. Global map (Courtesy of SAS).	148
6.9. Initial path for the primary rover (Courtesy of SAS).	149
6.10. Replanned path for PR and SR after successful traverse of the SR (Courtesy of SAS).	149
6.11. Panoramic picture of the Mars Yard (Courtesy of Airbus).	152
6.12. Representative Mission Scenario to exercise all the FASTER functions (Courtesy of Airbus).	153
6.13. Architecture for FASTER scenario with two collaborative rovers.	154
6.14. Flexible domain timelines for the FASTER Primary Rover. The highlighted boxes represent goals.	155
6.15. Hector Rescue Robot (Courtesy of SIM Group, TU Darmstadt).	157
6.16. Hector Subsystems.	158
6.17. Hierarchical behaviours included in the KDB of QuijoteExpress for USAR project.	159
6.18. Simulation of a rescue mission with QE and SE on-board a virtual Hector rover in the Gazebo simulator.	160
6.19. Hector robot in the SIM/TU Darmstadt arena.	161
6.20. User interface containing a representation of the timelines in execution for USAR project.	162

List of Tables

1.1. Autonomy Levels.	7
1.2. Impact of different planning techniques over three general properties of a planner: Plan Robustness, Performance and Expressiveness.	10
2.1. Features of <i>real-world</i> robotic scenarios	25
2.2. Allen's interval relations.	34
2.3. Comparison of uninformed algorithms.	40
2.4. Comparison of informed algorithms.	42
2.5. Planners comparison.	46
3.1. Combining HTN and APSI nomenclature in HTLN.	66
3.2. Correlation between hypergraph and HTLN elements.	70
3.3. Comparison of the number of relations in HTLN vs TRF.	87
4.1. New Autonomy Level.	99
4.2. Comparison of QEv2 and ASPEN.	100
4.3. Properties of synthetic rover domains for QEv1.	115
4.4. Properties of the synthetic rover domain for QEv2.	118
4.5. Results of QEv2-ST for Partial Ordered Problems.	120
4.6. Results of QEv2-MT for Partial Ordered Problems.	121
5.1. Format of the goal handler parameter used by RobCon.	127
5.2. Format of the ExecutePlan action used by SanchoExpress.	129
5.3. Format of the ExecutePlan action used by SanchoExpress.	131



List of Abbreviations

AEGIS	Autonomous Exploration for Gathering Increased Science
AI	Artificial Intelligence
AML	ASPEN Modelling Language
APSI	Automated Planning and Scheduling Initiative
ARC	Ames Research Center
ASE	Autonomous Sciencecraft Experiment
ASPEN	Automated Scheduling and Planning ENvironment
BFS	Breadth-First Search
BLS	Bridget Locomotion System
CAIP	Constraint-based Attribute and Interval Planning
CASPER	Continuous Activity Scheduling and Planning, Execution and Replanning
CBI	Constraint-Based Interval planning
CLARAty	Coupled Layer Architecture for Robotic Autonomy
CSP	Constraint Satisfaction Problem
CX-1	Citizen Explorer 1
DEM	Digital Elevation Map
DFKI	Deutsches Forschungszentrum für Künstliche Intelligenz
DN	Decision Network
DRC	Darpa Robotic Challenge
DRP	Downward Refinement Property
DS-1	Deep Space 1 Spacecraft
DSF	Depth-First Search
DTN	Disjunctive Temporal Network
EO-1	Earth Observation 1
ESA	European Space Agency
EUROPA	Extensible Universal Remote Operations Planning Architecture
FAF	Fewest Alternatives First heuristic

FASTER Forward Acquisition of Soil and Terrain data for Exploration Rover

FDIR Failure Detection, Isolation and Recovery

FTF Fewest Threats First heuristic

GNC Guidance, Navigation and Control

GSFC Goddard Space Flight Center

HST Hubble Space Telescope

HSTS Heuristic Scheduling Testbed System

HTLN Hierarchical Timeline Networks

HTN Hierarchical Task Networks

IA Interval Algebra

IDDFS Iterative Deepening Depth-First Search

IPC International Planning Competition

JPL Jet Propulsion Laboratory

KDB Knowledge DataBase

LM Landmark

MAPGEN Mixed Initiative Activity Planning Generator

MAV Mars Ascending Vehicle

MC Model Checking

mDCP motorized Dynamic Cone Penetrometer

MDP Markov Decision Processes

MER Mars Exploration Rover

MIMO Multiple-Input Multiple-Output

MISUS Multi-rover Integrated Science Understanding System

MMOPS Mars Mission On-Board Planner and Scheduler

MPF Mars Pathfinder

MSL Mars Science Laboratory

MSLICE Mars Science InterfaCE

MSR Mars Sample Return

MTF Most threats First heuristic

NASA National Aeronautics and Space Agency
OBC On-Board Computer
PA Point Algebra
PDB Pattern DataBase
PDDL Planning Domain Description Language
POMDP Partially Observable Markov Decision Process
POP Partial Order Planning
PR Primary Rover
PSI Phoenix Science Interface
QE QuijoteExpress mission planner
RAMS Reliability, Availability, Maintainability and Safety
RAX Remote Agent Experiment
RCOS Robot Control Operating System
ROS Robot Operating System
SAS Sample Acquisition System
SCL Spacecraft Command Language
SCR Sample Catching Rover
SD Sufficiently Decomposed
SE SanchoExpress executive
SFR Sample Fetching Rover
SHOP Simple Hierarchical Ordered Planner
SIPE System for Interactive Planning and Execution
SISO Single-Input Single-Output
SMA Simplified Memory-bounded A*
SR Scout Rover
STP Simple Temporal Problem
STRIPS Stanford Research Institute Problem Solver
T-REX Teleo-Reactive Executive
TCSP Temporal Constraint Satisfaction Problem

TLP TimeLine Planning
TRF Timeline Representation Framework
TSP Travel Salesman Problem
UCS Uniform-CoSt
UDS Unfold, Decompose and Schedule resolver strategy
UML Unified Modelling Language
USAR Urban Search and Rescue
USDS Unfold, Schedule, Decompose and Schedule resolver strategy
VTT Visual Target Tracking
WLSIO Wheel-Leg Soil Interaction Observation
WP WayPoint

Abstract

The transition of mobile robots from a controlled environment towards the *real-world* represents a major leap in terms of complexity coming primarily from three different factors: partial observability, non-determinism and dynamic events. To cope with them, robots must achieve some intelligence behaviours to be cost and operationally effective.

Two particularly interesting examples of highly complex robotic scenarios are Mars rover missions and the Darpa Robotic Challenge (DRC). In spite of the important differences they present in terms of constraints and requirements, they both have adopted certain level of autonomy to overcome some specific problems. For instance, Mars rovers have been endowed with multiple systems to enable autonomous payload operations and consequently increase science return. In the case of DRC, most teams have autonomous footstep planning or arm trajectory calculation.

Even though some specific problems can be addressed with dedicated tools, the general problem remains unsolved: to deploy on-board a reliable reasoning system able to operate robots without human intervention even in complex environments. This is precisely the goal of an automated mission planner.

The scientific community has provided plenty of planners able to provide very fast solutions for classical problems, typically characterized by the lack of time and resources representation. Moreover, there are also a handful of applied planners with higher levels of expressiveness at the price of lower performance. However, a *fast*, *expressive* and *robust* planner has never been used in complex robotic missions. These three properties represent the main drivers for the outcomes of this thesis.

To bridge the gap between classical and applied planning, a novel formalism named **Hierarchical TimeLine Networks** (HTLN) combining Timeline and HTN planning has been proposed.

HTLN has been implemented on a mission planner named QuijoteExpress, the first **forward-chaining timeline planner** to the best of our knowledge. The main idea is to benefit from the great performance of forward-chaining search to resolve temporal problems on the state-space. In addition, QuijoteExpress includes search enhancements such as parallel planning by division of the problem in sub-problems or advanced heuristics management. Regarding expressiveness, the planner incorporates HTN techniques that allow to define hierarchical models and solutions. Finally, plan robustness in uncertain scenarios has been addressed by means of sufficient plans that allow to leave parts of valid plans undefined.

To test the planner, a novel lightweight, timeline and ROS-based executive named SanchoExpress has been designed to translate the plans into actions understandable by the different robot subsystems.

The entire approach has been tested in two realistic and complementary domains. A cooperative multi-rover Mars mission and an urban search and rescue mission. The results were extremely positive and opens new promising ways in the field of automated planning applied to robotics.

Keywords: Automated Planning, Automated Execution, Mars Rover, Mobile Robot, Forward-Chaining, HTN, HTLN, Timeline Planning, Non-determinism, Uncertainty



Zusammenfassung

Der Übergang von beweglichen Robotern von einer kontrollierten Umgebung in eine reale Welt stellt einen großen Komplexitätssprung dar, der hauptsächlich auf drei unterschiedlichen Faktoren beruht: teilweise Beobachtbarkeit, Nicht-Determinismus und dynamische Ereignisse. Um diese zu beherrschen müssen Roboter ein gewisses intelligentes Verhalten entwickeln, um kostengünstig operieren zu können.

Mars Rover Missions und der Darpa Robotic Challenge (DRC) sind zwei besonders interessante Beispiele von hochkomplexen Roboterszenarien. Trotz wichtiger Unterschiede in Randbedingungen und Anforderungen besitzen beide Szenarien ein gewisses Autonomielevel, um spezielle Probleme überwinden zu können. Mars Rover zum Beispiel, wurden mit multiplen Systemen ausgestattet, um autonome Nutzlastoperationen zu ermöglichen und dadurch den wissenschaftlichen Nutzen zu erhöhen. Im Falle von DRC nutzen die meisten Teams autonome Schrittplanung oder autonome Berechnung der Bewegung des Roboterarms.

Obwohl einige spezielle Probleme mit eigens dafür entworfenen Tools bearbeitet werden können, bleibt das generelle Problem ungelöst: Die On-board-Installation eines zuverlässigen Entscheidungssystems, das in der Lage ist, Roboter ohne menschliche Intervention selbst in komplexen Umgebungen zu steuern. Genau dies ist das Ziel eines automatisierten Missionsplanungstools.

Wissenschaftler haben eine Vielzahl von Planungstools entwickelt, die in der Lage sind, sehr schnelle Lösungen für klassische Probleme zu liefern, die sich typischerweise dadurch auszeichnen, dass die Darstellung von Zeit und Ressourcen fehlt. Darüber hinaus gibt es eine Handvoll angewandter Tools mit tieferer Detailtreue, was aber zu Lasten deren Performance geht. Ein schneller, expressiver und robuster Missionsplaner wurde jedoch noch nie in einer komplexen Robotermision benutzt. Diese drei Eigenschaften stellen die Hauptmotivation dar für das Ergebnis dieser Arbeit.

Ein neuer Formalismus, Hierarchical TimeLine Networks (HTLN), wurde vorgeschlagen, der Zeitplan und HTN Planung verknüpft, um die Lücke zwischen klassischer und angewandter Planung zu schließen.

HTLN wurde in dem Planungstool QuijoteExpress implementiert. Dieses Tool ist nach unserem Wissen, das erste forward-chaining Zeitplanungstool. Die Idee ist von der hohen Performance der forward-chaining Suche zu profitieren, um zeitliche Probleme im Zustandsraum zu lösen. Zusätzlich enthält QuijoteExpress Suchverfeinerungen wie paralleles Planen durch Aufteilen des Problems in Unterprobleme oder fortgeschrittenes Management von Heuristiken. Bezüglich Expressiveness verwendet das Planungstool HTN Techniken, die erlauben hierarchische Modelle und Lösungen zu definieren. Zu guter Letzt wurde Planungsrobustheit durch das Konzept von "ausreichender Planung" adressiert, die erlaubt, dass Teile von gültigen Plänen undefiniert bleiben.

Um das Planungstool zu testen, wurde ein neues "light" Zeitplan- und ROS-basiertes Executable entworfen, um die Pläne in Aktionen zu übersetzen, die von den verschiedenen Robotersubsystemen verstanden werden können.

Der gesamte Ansatz wurde in zwei realistischen und komplementären Bereichen getestet. Eine kooperative multi-Rover Mission und eine urbane Such- und Rettungsmission. Die Ergebnisse waren extrem positiv und eröffnen neue vielversprechende Wege auf dem Gebiet von Automatisiertem Planen angewendet in der Robotik.

1 Introduction

Robotics represent a complex research field which still require multiple scientific advancements. Robots are supposed to be, at least to some degree, autonomous actors, otherwise, they should not be consider as team members but simply as tools. The last few decades have seen an increasing presence of robots in factories, but the transition from these structured environments to unstructured ones has been proved harder than expected.

One of such environments is space exploration. Space robotics show a big diversity including telescopes, landers, automated vehicles, satellites, humanoid robots, flying spheres, robotic arms or exploration rovers, all demanding some level of autonomy. Among all of them, planetary rovers are particularly challenging in terms of autonomy.

The Sojourner rover landed on Mars in 1997 demonstrated how important autonomy is. Sojourner was idle 40 to 75% of the time waiting for orders from Earth. In April 2009, Spirit Mars Exploration Rover (MER) got trapped on soft sand and could not get out again. After the landing in 2012, Curiosity was driven manually for more than one year with the increasing overhead for operators and decreasing science return it represents.

No other spacecraft missions are run so carefully as rovers. Even though several technical reports and publications claim for higher levels of autonomy on-ground and on-board rovers [57, 6, 76], few novelties are seen from one mission to the next in this regard and operations mostly depend on highly detailed plans generated on-ground. The main reason that makes rovers so unique is the strong interaction between the robot and a harsh, unstructured and dynamic environment such as Mars, which poses a high risk for the spacecraft in case any problem occurs. In addition, the high cost of rover missions calls for caution with respect to operations.

However, not all are bad news. The same factors presented in the previous paragraph can be claimed in favour of more autonomy. Indeed, the communication delays with Mars are such that teleoperation is unrealistic. Given the increasing complexity of future missions, the advantages in terms of science return and costs are undeniable. As an example, MER would have never been so successful without the AUTONAV system [17]. While autonomous navigation is broadly used, future missions will require higher levels of autonomy to fulfil their objectives. As an example, Mars Sample Return (MSR) mission will likely use a lightweight rover to recover a cannister with soil samples gathered by a previous mission and bring it back to an Earth return vehicle. This rover will use an unprecedented level of autonomy to increase its driving speed and perform opportunistic science during the long traverse towards the sample catching area. Opportunistic science and, more in general, automatic plan generation/repair on-board poses a lot of questions yet to be addressed, such as how to create robust plans for highly uncertain environments.

Another example demanding autonomous dexterous robotics is disaster scenarios. On March 11th 2011, the unfortunate events unleashed in Fukushima Daiichi nuclear power plant awoke the scientific community, showing the real deficiencies of present technology. Japan, supposedly a leader in the field of robotics, had none to send into Fukushima when the crisis began. Less than a week after the tsunami and earthquake, iRobot sent two robots, spare parts and engineers to the disaster area, but it took more than four weeks until they were used [82]. Other robots later on delivered from different countries (including Quince from Japan) experienced similar problems. Several factors such as limited communi-

cations, poor situation awareness or the complexity inherent to the operation of the platforms were part of the reasons for the poor performance displayed [113]. Even though a shared autonomy system would have significantly reduced the operator's workload, these robots were teleoperated because of the high complexity of the scenario and lack of trust from the customer.

Two competitions are of special relevance for disaster scenarios: the RoboCup Rescue League¹ and the DARPA Robotics Challenge².

In the former, the robots have to find the victims in an arena representing a building partially collapsed, determine their situation and location, and then report back their findings. Even though events like Fukushima showed that autonomy is advisable in these situations, at present all Rescue League teams rely on semi-autonomous systems with an operator in the loop. Once the robot starts a mission, it has pre-programmed behaviours to achieve the goals. If required, the operator can manually assign specific tasks to the robot, from high level such as find victim to low level such as manual driving.

Intended to be an answer to the Fukushima disaster, the DARPA Robotics Challenge (DRC) aimed to test some of the skills needed by robots in such scenarios. Due to different factors, specially the lack of time suffered in both the trials and DRC Finals, the teams had to focus on the development of sufficient basic robotic capabilities, leaving aside those non-mandatory such as autonomy. The level of autonomy used for the tasks was very low and most teams focused in control under human supervision where the operator performs the action in a virtual world and the movements are then sent to the "real" robot, with the exception of walking.

Rescue robots and planetary rovers actually present similar requirements in terms of autonomy:

- Need for autonomy: In many circumstances, teleoperating a robot is not possible. In field robotics, the presence of infrastructures or high levels of radiation might block the signal, while in space the long distances impose prohibitive delays in the communications.
- Robust plans: A plan should not fail in case it suffers slight deviations.
- Need to represent time: Most actions of a physical system take time to be executed. Moreover, sometimes one action requires others to be synchronously executed. These temporal constraints should be considered during planning.
- Dealing with resources: Physical systems such robots have resources, e.g. energy or memory. It is important to model and reason about them in order to generate realistic plans.
- Dealing with uncertainty: Uncertainty comes from three different dimensions: (1) Dynamic environment that can unexpectedly change; (2) Partial observability of the world surrounding the robot; and (3) Unstructured environment makes it impossible to precisely ascertain the outcome of the robot actions.

In spite of these similarities, the criticality of both scenarios is quite different. First, there is no possible real-time communication with Mars robots as the round trip signal takes between 8 to 42 minutes. Besides that, eclipses either for direct to-Earth communications or with an orbiter might prevent any contact at all. On Earth, even in very complicated situations such as Fukushima where radiation and heavy shielded walls represented a big problem, high power WIFIs and/or wired communications might solve the situation. Second, the cost of rover missions, around 2.5 billion USD for Mars Science Laboratory (MSL), together with the impossibility of recovering the robot in case of error plays an important

¹ <http://www.theroboticschallenge.org/>.

² <http://www.robocuprescue.org/>.

role. Whereas it is affordable to lose a rescue robot, as it happened with Quince, there is no room for mistakes in the space sector. Third, time is more critical for rescue robots; the first three days after the onset of a disaster has been dubbed the “72-hour golden rescue period”, after which the survival rate sharply declines. On the other hand, improving operations performance is more a benefit than something critical for planetary rovers.

It is clear then that autonomy provides multiple advantages. In order to narrow down that concept, the ECSS Space Segment Operability Standard[119] provides a formal definition of autonomy that can be easily extrapolated to rescue robots:

Definition 1 (On-board autonomy) *On-board autonomy management addresses all aspects of on-board autonomous functions that provide the space segment with the capability to continue mission operations and to survive critical situations without relying on ground segment intervention. The implementation of on-board autonomy depends on the specific mission requirements and constraints, and can therefore vary between a very low level of autonomy involving a high level of control from ground to a high level of autonomy, whereby most of the functions are performed on-board.*

Level	Description	Function
E1	Mission execution under ground control	Real-time control from ground for nominal operations. Execution of time tagged commands for safety issues
E2	Execution of pre-planned, ground-defined, mission operations on-board	Capability to store time-based commands in an on-board scheduler
E3	Execution of adaptive mission operations on-board	Event-based autonomous operations. Execution of on-board operations control procedures
E4	Execution of goal-oriented mission operations on-board	Goal-oriented mission replanning

Table 1.1.: Autonomy Levels.

This same document classifies autonomy in different levels, shown in Table 1.1. The question is then which is the appropriate level of autonomy. While some scenarios like Robocup Soccer or the DARPA Grand Challenge mostly demand reactive systems, others like RoboCup Rescue, DRC or Mars rovers also require deliberative behaviours. Spirit and Opportunity were based on a E3 autonomy level, claimed to be the minimum level for a planetary mission [162], been E4 desirable.

In order to achieve E4, automated planning on-ground and at some extent on-board is required to generate and modify plans. Besides, an executive is required on-board to perform the activities defined in the plan. Although automated planning represents the main field of study, execution also presents multiple challenges that have been explored in this thesis.

1.1 Automated Planning: from Theory to Practice

Humans act more frequently than they plan. Generally speaking, explicit deliberation only happens in dangerous or new situations in which the task to perform is complex [12].

Planning has been an active research topic in artificial intelligence and presents nowadays several forms including mission planning, path planning, motion planning, etc. This thesis focus on *domain-independent* solutions able to be easily reused in different scenarios like those presented above. Before getting i

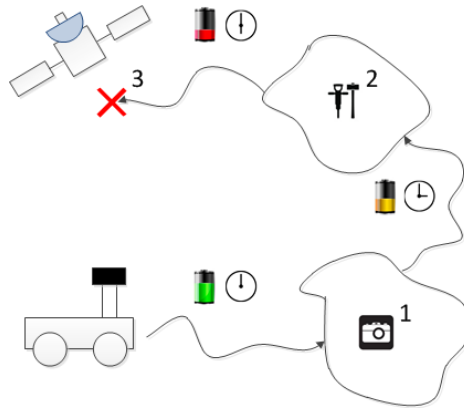


Figure 1.1.: Mars rover exploration problem.

Definition 2 (Automated planning) *Planning is the reasoning side of acting that aims to organize actions according to their expected outcomes in order to achieve some given goal [81]. Automated planning is the area of Artificial Intelligence (AI) that studies this process.*

A planner typically receives two inputs. One is a domain containing a formal description of both the agent executing the actions (the robot in our case) and the environment surrounding the agent. The second is a problem describing the present state of the agent and the goals to achieve. The problem of plan generation in AI consists on computing a sequence of actions or states (the plan) to transform the initial state of the world into another that satisfies the set of goals with limited (or no) human intervention.

An example of a problem is depicted in Figure 1.1. A rover must navigate through unknown terrain towards an area (labelled as 1) where it has to take some pictures. Later on, it moves to another area (2) where it performs some science such as chemical analysis of rocks. Finally, it moves to the final position for that sol³ and communicates the information to an orbiter. During the process, apart from determining the sequence of activities or states, the planner must have into account the resources such as battery and the time required for each action. In case that the execution diverts from the original plan, it must be cancelled and some counter-measure such as on-board replanning or wait Earth for instructions should be taken.

One informal division in the field of automated planning can be done between theoretical and applied planners, even though there are several examples laying in the middle.

Theoretical planning focus to a great extent on improving the performance of planners. Most modern planners are based on heuristics, pieces of software telling the planner which nodes of the search space should be analysed first. As the main planning branch, the mathematical foundations have been well studied, providing a strong theoretical background used to define several “de facto” standards broadly used by the community. For example, almost every modern classical planner is based on any of the multiple versions of the Planning Domain Description Language (PDDL) [104, 69]. Several synthetic domains have been built to test the performance based on well-defined metrics. The bi-annual International Planning Competition (IPC) [150] represents the main event to understand the state-of-the-art on planning, divided in different subtracks that cover the main research lines, namely sequential (subdivided in optimal and satisficing), multi-core and temporal. After the competition, the source-code must be shared in order to disseminate the knowledge and improve future planners.

³ sol is the name given to Martian days.

Conversely, the world of applied planning is by nature less organised. Companies use to be reluctant to freely share code, there are no standard languages, domains or competitions whatsoever. As planners are oriented to solve specific-tasks, many are domain-dependent, making it harder to use the experience earned in other problems, or to apply algorithms coming from the theoretical world.

In consequence, there is a gap between the two research areas. On the one hand, most of the theoretical systems cannot be used for real-world problems as they are not expressive enough or they lack specific knowledge required to solve the task. On the other hand, applied systems do not support standards and do not promptly adopt new ideas coming from the theory resulting in low performance. In this thesis we try to bridge this gap by evolving the novelties shown in modern theoretical planning systems into a new temporal planner.

1.2 Motivation and Contributions

The primary motivation of this thesis is to provide a **novel planning approach** based on the Advanced Planning and Scheduling Initiative framework or APSI (see 2.4.4) able to demonstrate high levels of autonomy (on-ground and on-board) for planetary exploration missions such as MSR. Nevertheless, the strong similarities with other scenarios such as rescue robots facilitated the extension of the scope to the broader unmanned autonomous robotics in partially observable, non-deterministic and dynamic scenarios. This work also strives for a complete system design incorporating the mission planner and an executive to conduct experiments and validation with real world robots.

Most of the requirements presented at the beginning of the chapter, further extended in Section 2.2.2, have been already achieved by different state-of-the-art mission planners. It is possible nowadays to represent time and resources or compute solutions efficiently. However, focusing on temporal planning, there are three fundamental capabilities yet to be completely addressed by a planner intended to be used in these scenarios:

- **Robustness:** It is not possible to have all the relevant information about the robot or the surrounding environment in uncertain scenarios . In consequence, plans tend to fail and repetitive replanning becomes essential, yet it is still required to achieve the goals. In order to create more robust plans with respect to deviations during execution, it is preferable to use flexible plans that are filled up progressively as the information arrives. This aspect is of utmost importance when the mission or system is critical, as is the case of a Mars rover.
- **Performance:** Enables the planner to be used in situations where responsiveness plays an important role as it happens in disaster scenarios, where the chances to survive rapidly decrease with time. Planning/replanning activities should be fast in order to resume operations as soon as possible. In this thesis, several techniques such as parallel planning and classical planning approaches are proposed to improve the planner performance.
- **Expressiveness:** Allow users to model domains and problems closer to the reality and at the same time easier to understand. To achieve it, the use of hierarchies has been added to the traditional representation based on automata to model domains and robot behaviours.

This thesis proposes a number of techniques listed in Table 1.2, each oriented to tackle one or more of the three previous points.

Hierarchical Timeline Networks theory [170] (*HTLN*) combines Hierarchical Task Networks (*HTN*) and *timeline* planning. *HTLN* addresses the problem of robustness as it allows human experts to in-

Feature	HTLN	Sufficient Plan	Parallel Planning	Heuristics
Robustness	✓	✓		
Performance	✓		✓	✓
Expressiveness	✓			

Table 1.2.: Impact of different planning techniques over three general properties of a planner: Plan Robustness, Performance and Expressiveness.

roduce knowledge in the form of domain-dependent, highly robust plan fragments. In terms of expressiveness, HTLN helps to generate hierarchical models and plans, easier-to-understand by humans. Finally, the use of HTN planning represents an improvement in performance thanks to the insertion of plan fragments in one single step.

Sufficient plan allows the planner to generate partially defined plans in situations where the knowledge about the environment and the required actions to achieve the goals is limited. The intention is to prevent plan failures derived from early assumptions and therefore to provide more robust plans in uncertain scenarios.

The *Parallel planning* technique presented in this thesis is based on a highly effective way to divide a problem into subproblems, which takes advantage of certain properties of the hypergraph structure used in HTLN to represent problems. Subproblems can be then planned in parallel preserving validity and optimality with the resulting performance improvement.

Finally, a number of search algorithms such as A*-style borrowed from the classic-planning world have been also used to improve the performance.

With this building blocks we have implemented a state-of-the-art planning system called QuijoteExpress [174, 167, 175]. We expect QuijoteExpress to be able to satisfy the requirements and to offer the capabilities mentioned above, that is, to generate plans for complex problems with partial information in uncertain environments in a robust and agile manner. To prove it, the planner has been extensively tested in two different scenarios: a Rescue-robot and two Mars rovers, hereafter referred to as the *reference scenarios*.

1.3 Outline of the Thesis

As mentioned before, this thesis lies between the theoretical and applied world. Due to its multidisciplinary nature, the background covers a wide range of topics including classical and applied planning, plan execution and (space) robotics. Next, a novel planning formalism called HTLN is described. The main output of the thesis, a planner based on HTLN and an executive able to directly manage the planner output are also presented and their novelties analysed. Finally, their performance is evaluated and final conclusions extracted. Below, we outline the chapters in more detail.

Chapter 2: Background

It is split in two main areas: planning and execution. Regarding planning, the chapter starts with an analysis of the state-of-the-art in classical planning giving special emphasis to *forward-chaining heuristic search*. Next, a similar overview on applied planning is conducted focusing on Hierarchical Task Networks (HTN) and temporal planning. The combination of HTN, temporal planning and forward-chaining represents one of the central ideas of the thesis. A section is also dedicated to APSI to settle down the formalism in which this work is based. Regarding execution, the main features of an executive

are identified and an overview of the state-of-the-art executives is presented. The chapter is closed with the benchmarking metrics used to measure the different properties of the planner and executive.

Chapter 3: Hierarchical Timeline Networks: A Novel Planning Formalism

This chapter presents the Hierarchical Timeline Networks (HTLN) theory, one of the cornerstones of this thesis. First, we give an overview of hypergraph theory, used to represent problems and solutions in QuijoteExpress. Next, Hierarchical Task Networks (HTN) and temporal planning theory is studied, both of capital relevance in the conception of HTLN theory. Finally, HTLN theory and its consequences in the planning system are described.

Chapter 4: QuijoteExpress Planner: A Novel Planning System

This chapter focuses on the specific implementation of HTLN into one planner. QuijoteExpress is a domain-independent, heuristic-based, parallel, hierarchical temporal planner that combines the novelties presented in HTLN and classical planning techniques. We have pursued the creation of a new planner that stands out in three different areas: (1) Strong performance thanks to the use of informed search algorithms and HTLN; (2) More flexibility thanks to the use of heuristic search; and (3) More expressive power thanks to the use of HTLN. Each novel feature in timeline planning such as parallel planning, multi-heuristic evaluation, deferred heuristic evaluation, etc. are individually studied and the impact in the global performance analysed. From an algorithmic point of view, the chapter presents the search algorithms and heuristics developed for QuijoteExpress.

Chapter 5: SanchoExpress: Flexible Execution with FDIR Capabilities

Once the plan is generated, it needs to be executed. A number of executives are analysed, none of which fulfils the requirements for the Mars and Rescue scenarios. SanchoExpress is a timeline-based executive in charge of dispatching the appropriate commands to achieve the tasks in the timelines, at the appropriate time frame and to the appropriate robot subsystem. It also monitors the execution and reports back to the planner in case of errors. SanchoExpress can execute any plan represented with the appropriate timeline format regardless of the planner used to generate it. Moreover, it is domain-independent, that is, it can be used for any kind of robot without having to modify it. In this chapter, the main concepts, architecture and implementation details of SanchoExpress are presented and compared to other already existing executives.

Chapter 6: Evaluation in Real-World Scenarios

One of the differences between synthetic scenarios such as the blocks-world and “real” ones is that there might be significant deviations between the plan generated and the outcome of the execution. Previous chapters are intended to demonstrate the soundness of the different novelties from a theoretical point of view. This chapter changes course and test the planner as a whole in two scenarios. The first, named FASTER (Forward Acquisition of Soil and Terrain data for Exploration Rover) represents a Mars exploration mission with two rovers that collaborate to traverse faster and safer towards a target. The second, named USAR (Urban Search and Rescue) involves a rescue robot that must find the victims in a disaster scenario and assess their state. Both scenarios helped to demonstrate high levels of autonomy with real robots in complex environments.

Chapter 7: Conclusions

Last chapter provides an overview of what has been achieved highlighting the main contributions of the thesis and the remaining open points for future research topics.



2 Automated Planning and Execution: Background

This chapter covers a number of key topics for the thesis, including automated planning, execution, robotics and space.

First of all, the most relevant planning formalisms are identified and characterized in order to analyse which are the most relevant for the scenarios presented in Chapter 1. A number of synthetic and real domains are then presented. Next section presents a list of relevant frameworks and planners with special emphasis on HTN and Timeline-based planners. Following section analyses state-of-the-art executives. Some metrics and relevant properties for planners and executives are then presented and finally, the conclusions summarize what is missing in the field of autonomy for mobile robots in complex environments and what must be done to tackle the problem.

2.1 Classical vs. Applied Planning

Planning is the reasoning side of acting that aims to achieve some predefined objectives and Automated planning is the area of Artificial Intelligence that studies this process.

There are several forms of planning, traditionally subdivided in two major groups: classical (or theoretical) and applied. The gap between the two of them is a reiterative topic in the main planning and AI conferences. All planning systems used for the scenarios presented in Chapter 1 are based on some kind of temporal planning, mainly timeline-based. However, it is clear that they could be highly benefited, specially in terms of performance, from the novelties brought by modern classical planners.

In fact, both planning styles share multiple concepts. Most classical and applied planners have a search space (see definition 8) containing at least one list (some have several) of candidate solutions that the solver needs to evaluate. All types of uninformed and informed search are also shared between the two worlds, with a predominant use of A*-style algorithms, and rely on heuristics based on the same principles even though the type of knowledge used differs.

However, there are some key differences that make the application of classical planners to applied domains complicated. These differences, analysed in [142], can be grouped as follows:

Knowledge about the Domain (time, resources)

The representation of time and resources is critical for applied systems. Moreover, all planners analysed for the space domain are based on timelines and contain schedulers to manage resources. On the contrary, most of the theoretical planners do not support any of them. Among the few theoretical temporal planners, only a minority support the whole specification of PDDL 3.1 which is already less expressive than the applied temporal planners presented in Section 2.4.4, specially in terms of temporal relations. With respect to the resources, supporting them require numerical fluents that imply infinite domains making planning undecidable [43, 71, 35].

Knowledge about “Good Plans” (oversubscription, optimization, robustness, flexibility)

Timeline planners facilitate the reasoning about plan quality, expressed as hard (need to be enforced) or soft (preferences) constraints. Quality metrics are defined to assign a score to the plan that the planner tries to increase following the constraints [43, 123].

Search-Control Knowledge

Interesting problems might present time and memory complexities exponential to the size of the problem, making blind search ineffective. In consequence, planners require some kind of search control to lead them towards the most promising areas of the search space. Classical planners typically rely on domain-independent heuristics and applied planners on domain-dependent. Domain-dependent knowledge can dramatically improve the planner performance and sometimes make the difference between solving a problem in exponential or polynomial time (e.g. [83, 141]), or allowing to solve problems orders of magnitude more complicated (e.g. [115, 117, 4]).

Cooperative Planning

Sometimes, it is difficult to formally specify all constraint and operation preferences for a given domain, making a fully automatic plan generation approach inapplicable. Nonetheless, the complexity of the problem also prevents to generate plans manually. Consequently, many timeline-based planning systems adopt a **mixed-initiative** approach to support collaboration between humans and an planners to build a high quality activity plan [24, 21, 22].

Ready for Execution

Applied planners produce plans usually intended to be executed by physical systems. In unstructured domains, unforeseen events such as faults or science opportunities require fast response. In such situations, a monitoring system must alert the planner about the new situation in order to generate a new plan [40] by means of fast replanning / rescheduling.

General Assumptions about the Domain and Problem

Classical planning is based on a number of assumptions, presented in Section 2.2.1. As these assumptions get relaxed, planning complexity increases to a point where it becomes intractable.

Search space

In [50], the strengths and weaknesses of state-space and plan-space approaches are evaluated. A key advantage of state-space is that a planner based on this approach performs search based on a complete understanding of the state of the world and therefore it is possible to design more informed heuristics to guide the process. Moreover, as it always plans forward, it avoids the need to resolve threats, as is the case in Iterative Repair, by imposing a total-order on actions: each new action added to the plan comes after all those already in the plan. The total ordering of actions ensures that each new action cannot threaten earlier constraints (it is automatically promoted), and no later action can threaten its constraints. On the other hand, the price to pay is that actions never occur in parallel and their ordering is early determined, which might lead to poor-quality plans and even make search more difficult.

Use of “de-facto” Standards

Among other factors, PDDL language and the International Planning Competition (IPC) have played a mayor role in the standardization of classical planning.

PDDL (see Section 2.3.1) the reusability of code, models, etc. and to easily compare different planners[69].

Regarding the International Planning Competition (IPC), its objectives are to provide an empirical comparison of planning systems, to highlight challenges to the community in the form of problems at the edge of current capabilities, to propose new directions for research and to provide a core of common

benchmark problems and a formalism that can aid in the comparison and evaluation of planning systems. The rules of the competition are clearly stated:

- Language: Planners must use a specific PDDL version.
- Tracks: Each planner can take part on one of the following tracks: sequential, temporal and preferences.
- Evaluation: Metrics are defined to evaluate planners based on plan quality and solving time. Evaluation of each planner is performed in a standard platform and has a predefined time to solve all the tasks.

Summarizing, applied planners are very expressive (supporting time, resources, external procedures), effective on solving specific problems and can be easily tailored with external procedures and control knowledge, partially thanks to its less restrictive formalisms. On the other hand, classical planners clearly outperform applied planners in solving general problems, are based on well defined standards and have a strong research community that constantly produces novelties.

2.2 Planning Domains

As a consequence of the differences analysed in the previous section, each planning technique is specially suitable for some specific type of domains. The following subsections analyse the characteristics and assumptions of classical and applied domains, used later on to classify problems and identify relevant solving techniques.

2.2.1 Classical Domains

Classical domains are used in academia with scientific purposes. They typically represent abstractions of real problems based on a number of assumptions referred to as the **restricted model** [81]:

- A0 (Finite Σ): The system Σ has a finite set of states.
- A1 (Fully observable Σ): The system Σ is fully observable, i.e., one has complete knowledge about the state of Σ .
- A2 (Deterministic Σ): Σ is deterministic, i.e, for every state s and every action μ , there is a transition function $|\gamma(s, \mu)| \leq 1$. If an action is applicable to a state, the transition leads to another single state.
- A3 (Static Σ): Σ has no internal dynamics, staying in the same state until an action is applied.
- A4 (Restricted goals): The planner handles only restricted goals specified as goal states S_g . Extended goals such as states to be avoided or utility functions are not considered.
- A5 (Sequential plan): A solution plan to a planning problem is a linearly ordered finite sequence of actions.
- A6 (Implicit Time): Actions and events have no duration as time is not represented in this model.
- A7 (Offline planning): The planner is not concerned with any external change that may occur while it is planning; it plans for the given initial and goal states regardless of the current dynamics, if any.

Specially relevant is the International Planning Competition (IPC) which tests the planners against a number of problems updated in every edition in order to cover new research topics. The domains of the IPC are organized in two groups: Temporal and Sequential, with the later one including domains for the satisfying, agile, optimal and multi-core tracks.

Several domains are submitted every IPC. The selection criterion is based on the following principles:

- Reproducible: Others can follow the method and produce the same (sort of) problems.
- Without bias: Does not favour a system against another.
- Useful: not resulting in all problems solved trivially, or all unsolvable.
- General: It can be applied to any set of planners, on any domain.

Different domains help to focus on different planning problems such as action-cost, negative-preconditions, conditional-effects or durative actions. However, in opposition to real domains, they can be simplified to a level in which planners are able to solve them.

One specially hard domain in the IPC-2014 sequential track was Visit-all [102]. An agent in the middle of a square grid $n \times n$ must visit all the cells in the grid. This is an extremely simple problem to solve non-optimally as it is just the result of multiple easy but conflicting goals that can often be achieved trivially, once they are serialized. However, this domain is very difficult for “pure” heuristic search planners, mainly those based on delete-relaxation heuristics.

With respect to the temporal track, the Satellite domain is specially interesting for this thesis. One or more satellites are used to make observations, collecting and downlinking data to a ground station. The satellites are equipped with different sets of instruments, each with different characteristics in terms of appropriate calibration targets, data productions, energy consumption and requirements for warming up and cooling down. The satellites can be pointed at different targets by slewing them between different attitudes. There can be constraints on which targets are accessible to different satellites due to occlusion and slewing capabilities. Instruments generate data that must be stored on the satellite and subsequently downlinked when a communication window opportunity (which is fixed) opens with a ground station. Due to the high volume of data, it is not possible to download the whole storage in one single pass, therefore downlinks must be scheduled taking into consideration the activities related to production of data, storage capacity and available communication windows.

In the real problem there are additional difficulties such as the management of energy, the use of solar power, maintenance of operational temperatures during periods in shadow, targets are only visible during particular time-windows and the downlinking windows are variable. Representing these facts in PDDL2.1 is possible but not straightforward, remaining an area in which there is need for development. The two domains used to evaluate QuijoteExpress go in this direction, demanding more expressiveness than in those domains analysed so far, even though some simplifications are also required.

2.2.2 Applied Domains

In the case of applied planning, each specific domain implies its own specific assumptions. In the Mars rover and rescue robot scenarios, none of the assumptions in Section 2.2.1 are valid:

- A0 (Finite Σ): The use of numerical state variables (such as power, memory) and parameters (speed, position) implies the existence of infinite states, even if the system is modelled with finite state machines.

- A1 (Fully observable Σ): Some aspects of the state of the world are unknown, which is one of the causes of uncertainty for the reference domains. It has three consequences: (1) Planning based on a complete understanding of the world is not feasible; (2) Some of the assumptions considered during planning might be wrong; and (3) New relevant information for the plan might be discovered only during execution time.
- A2 (Deterministic Σ): Σ is non-deterministic, i.e., each action can lead to different possible states but, contrary to stochastic systems, no probabilities are attached to them. In consequence, A5 assumption should also be relaxed, as the plan must encode ways for dealing with alternatives, e.g., conditional constructs like “do a and, depending on its result, do either b or c” and iterative constructs like “do a until a given result is obtained”. Notice that the controller has to observe the state s in a closed-loop control. If assumption A1 is also relaxed, this leads to another difficulty: the controller does not know exactly the current state s of the system at run-time.
- A3 (Static Σ): Σ could spontaneously change its state due to external events. This represents the third factor of uncertainty for the reference scenarios.
- A4 (Restricted goals): Plans do not consist on reaching a final goal state, but rather on achieving a set of states during a given time. Moreover, it could be desirable to specify quality metrics to optimize the plan.
- A5 (Sequential plan): The plan might be a flexible structure where the execution of some parts depend on others.
- A6 (Implicit Time): Timeline-based planners consider durations for the actions and for the relations between them.
- A7 (Offline planning): In real-world scenarios, the plan execution must be checked online for possible deviations. In cases something unexpected happens, the executive must notify the planner in order to repair the plan.

These assumptions represent a big change with respect to the previous restricted model, making planning undecidable.

Space Domains

Due to the ever increasing complexity of spacecraft and missions, space agencies have started to move toward autonomous operations for both on-ground and on-board segments.

NASA and ESA represent with no doubt the reference point in terms of autonomy. The NASA New Millennium Program strove to build "faster, better and cheaper" missions with spacecraft autonomy becoming one of the crucial technologies to achieve this vision while ESA Proba family has launched several satellites dedicated to demonstrate on-board autonomy.

Specially relevant for this thesis are the rover missions. A **rover** (see Figure 2.1) is a type of space exploration vehicle designed to move across the surface of a planet or other celestial bodies. It is typically equipped with a wheeled locomotion system to move across hazardous terrain, although legged versions are common in research labs. Some rovers have been designed to transport astronauts like those used in Apollo missions while others are robots with different levels of autonomy depending on the mission constraints.

The hardware of a exploration rover is divided in subsystems that can be classified in two main groups:

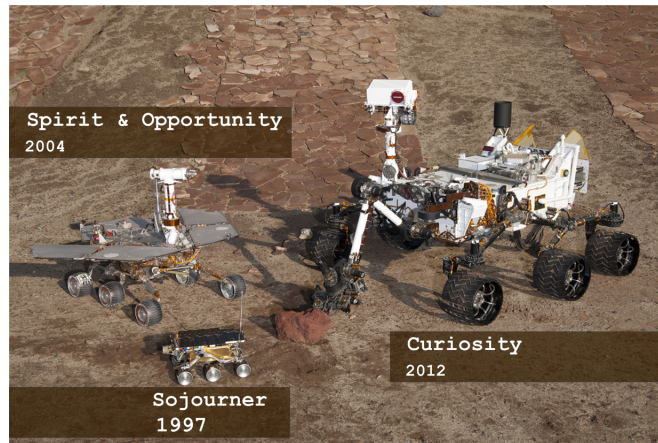


Figure 2.1.: Sojourner, MER and Curiosity Mars rovers (Courtesy of JPL).

- Platform: Involves the systems not related to scientific activities. Some examples are the GNC (Guidance, navigation and control), communications, mast, battery, etc.
- Payload: Represents the scientific hardware. The sample catching rover of MSR for example should have a vision system to analyse the rocks and a Sample Acquisition System (SAS) to store the most interesting rocks.

While the platforms are becoming more or less standard, the payload significantly changes between missions. This is the case for example of the 2020 Mars mission, that will largely reuse the Curiosity rover platform.

Reasons for Autonomy

The degree of autonomy depends on the mission requirements, including: Execution and monitoring of planned actions; Failure detection, isolation and recovery (FDIR); Automated planning/replanning/scheduling of activities; and Expert systems.

Regarding automated planning, in the traditional approach to spacecraft operations, humans on ground carry out a large number of functions including planning activities, sequencing spacecraft actions, verifying the spacecraft's state, recovering failures, etc. This approach seems to be unfeasible in future missions due to a number of reasons [41, 162]. The following is a non-exhaustive list of challenges faced by space missions that can be (at least partially) mitigated with on-board autonomy:

- Mission complexity: Space missions have increased their complexity exponentially [57, 6]. As a consequence, engineers need the help of automated tools to create better activity plans.
- Uncertainty: The sources of uncertainty exposed above (dynamism, unstructured and non-determinism) usually make plans to fail, resulting in the need to replan. In the best case, replanning imposes a huge workload to the personnel on-ground. In some missions however it might not be even possible on-ground (see next point).
- Limited communications: Some missions, specially in deep space, do not allow continuous or real-time communications with the robot for different reasons: (1) Eclipses preventing direct-to-Earth communications, specially relevant for surface operations, combined with limited communication windows with relay satellites; (2) Long-delays in the round-trip of radio signals, which in the case of Mars range between 8 and 44 minutes; (3) Limited availability of the Deep Space Network, which only allow few downlinks a day for each mission;

-
- Limited on-board resources: There are several examples of such limitations that have been resolved with more autonomy. For example, the limited on-board memory of Mars Express was solved by means of the Mexar system[34, 32]; The high power consumption of the antennas in MER was among the reasons to add on-board autonomous science detection in MER[28].
 - Science: Maximizing science return is always critical. Taking advantage of on-board planning, it can be achieved by means of opportunistic science[67], autonomous instrument placement[85], on-board data processing[28], etc.
 - Safety: Space represents a hazardous environment, as demonstrated with the loss of Spirit after getting stuck in a sand trap. Automated planning on-board simplifies the self-monitoring, on-board fault-management and health of the spacecraft.
 - Cost: Spacecraft operations are expensive. Long term missions like MER, active in Mars since 2004, represent a challenge for the operations team: producing day-to-day command sequences is a very demanding task that requires a huge team of experts. Automated tools on-ground such as MAPGEN[22] (Mixed Initiative Activity Planning Generator) helped engineers to create valid plans with less effort.

All these factors are specially critical for surface missions such as Mars rovers. As an example, the Sojourner rover was idle 40 to 75% of the mission time, waiting for orders from Earth. This situation has improved in modern rover missions such as MER or MSL thanks to on-board autonomous software such as Autonav (autonomous navigation).

In the new model of operations, the scientists will communicate high-level science goals directly to the spacecraft, which will automatically generate on-board the corresponding plan, verify its correctness and ultimately execute it without routine human intervention. In the presence of errors, the spacecraft will have to understand their impact and replan in light of the new information.

Next, a list of some of the most relevant missions from NASA and ESA in terms of autonomous operations is presented.

Hubble Space Telescope (HST)

The NASA/ESA Hubble Space Telescope, launched in 1990 has revolutionised modern astronomy, but also represented a cornerstone in terms of automated planning for space applications.

To cope with the tight short-term observing schedules, a framework named HSTS (Heuristic Scheduling Testbed System) was used for the first time to develop integrated P&S applications. The observation scheduler scaled up so well that in the end it was also used to plan long-term plans with all necessary activities such as instrument reconfiguration, telescope re-pointing, data communication, etc.

HSTS can be considered the father of modern temporal planners, at least in the space domain. The framework allows to represent the world as a set of **state variables** which vary over time. It also describes the dynamics of the domain by defining activities with pre and post conditions and the modelling of resources, another key concept in modern planners. Solutions are presented in a flexible manner as temporal constraint graphs, avoiding the problems of over-commitment inherent in “fixed times” scheduling frameworks. Finally, HSTS provides a uniform view of P&S processes as an iterative constraint posting process allowing a range of problem solving strategies (e.g. forward simulation, backward chaining, etc.). Most of these concepts are fundamental in modern temporal planners such as ASPEN, EUROPA or APSI, analysed in Section 2.4.4.

Deep Space One (DS-1)

Launched in 1999, it had the Remote Agent Experiment (RAX) on-board [110, 93], a layered agent architecture developed at NASA Ames Research Center (ARC) and the Jet Propulsion Laboratory (JPL). RAX represented a new model of operations where scientists communicated high-level science goals to the spacecraft converted by RAX into a plan executed and monitored on-board. The autonomous agent architecture comprised three modules: the constraint, timeline-based “Planning and Scheduling System” in charge of generating the activity plan; the “Smart Executive”, responsible for plan execution, local recoveries and temporal adjustments according to the actual conditions of executions; the model-based "Mode Identification and Recovery System" (Livingstone) in charge of monitoring the progress of execution from abstracted sensor data, diagnose the cause of off-nominal behaviour and suggest mode reconfigurations to be executed by the Smart Executive without any intervention from ground controllers.

Earth Observation One (EO-1)

Launched in 2003, the satellite was endowed with the Autonomous Sciencecraft Experiment (ASE) [44, 136]. EO-1 was developed and is operated by NASA Goddard Space Flight Center (GSFC), while ASE was developed by the Jet Propulsion Laboratory (JPL). ASE is composed of three components: Continuous Activity Scheduling and Planning, Execution and Replanning (CASPER), which is an on-board deliberative, model-based planner and scheduler; Spacecraft Command Language (SCL), which acts as a middleware execution engine; and the Science processing software to perform additional science classification and processing. There is also a complementary ground software called Automated Scheduling and Planning Environment (ASPEN), which works collaboratively with CASPER to generate plans. ASE differs from RAX in two aspects: Autonomous spacecraft control with continuous planning rather than batch-driven commands and the use of accurate and effective science models to achieve science-driven autonomy. ASE was a complete success and some of the tools such as CASPER and ASPEN are broadly use nowadays in several NASA missions (see 2.4.4).

Project for Onboard Autonomy (PROBA)-1

The Proba family is a series of micro-satellites intended to demonstrate different autonomous systems. Proba-1 was provided with automatic functions for both the on-board and the ground segment, including nominal operation and resource management, automated camera pointing and scanning, scheduling and execution of payload operations, data communication management or pass operations [147].

Mars Pathfinder (MPF) / Sojourner

The Sojourner rover of the Mars Pathfinder (MPF) mission, landed on Mars in 1997, was the first rover to demonstrate some level of autonomy. It drove a total of 84 meters during its entire mission. However its autonomy was limited because of concerns about reliability and verifiability. The rover could only execute rigid commands sequences rigorously checked and verified by mission control prior to being uploaded. The default response to unexpected behaviours was to abort the sequence and wait for the next communications opportunity. This approach resulted in a considerable fraction (between 40 to 75%) of the nominally 90 days of surface mission being used to determine the state of the remote system and return it to productive operation. Sojourner demonstrated the potential for robotic Mars exploration but at the same time showed the need for a more robust rover autonomy [156].

Mars Exploration Rover (MER)

Next rover mission, the Mars Exploration Rover (MER) that landed two rovers named Spirit and Opportunity on Mars in 2004, greatly extended the capabilities of Pathfinder/Sojourner, from 10s of

meters around its lander to many kilometres over different types of terrain. The efficiency of the MER mission was far better than in Pathfinder, thanks to a great extent to the autonomous navigation software (see 2.4.5).

MER operations are divided in strategic (time horizons about two weeks) and tactical (one sol time horizon). The major steps of the tactical process are [106]: (1) Downlink reception; (2) Science activity planning; (3) Activity plan refinement and validation; (4) Activity plan review; (5) Command sequence generation; (6) Sequence integration and validation; (7) Command review; (8) Transmission of commands to the spacecraft.

As MER did not have any on-board replanning capability, they are referred to as “semi-autonomous”. They have event-driven pre-programmed behaviours for many critical tasks as well as time-driven for some payload functions. However, several updates along the years of operations after the nominal mission brought increasing levels of autonomy both on-ground and on-board (see Section 2.4.5).

Mars Science Laboratory (MSL) / Curiosity

MSL with its rover Curiosity landed on Mars in 2012. Even though Curiosity is the most advanced rover ever made, it does not represent a revolution in terms of autonomy, at least during the nominal mission. The main effort has been oriented towards the consolidation of the software inherited from MER [154] and Phoenix missions, which will represent a standard platform in future missions such as Mars Sample Return (MSR). After more than one year of manual driving, the rover got a software update in 2013 including the Autonav system that allows for autonomous navigation and AEGIS for the ChemCam spectrometer autonomous target selection [65].

ExoMars

ExoMars 2020 represents the first rover mission of ESA. The ambitious requirements in terms of number of experiment cycles per sol for the Humboldt(lander) and Pasteur(rover) payloads, including operations such as instrument placement, sample acquisition using a driller or sample processing, and for the rover mobility, with distances between cycles in the range of 500 meters, suggest an autonomy level equal or higher to MER with the probable addition of on-board planning/scheduling [162] and multi-sol operations [138]. Some studies about possible on-board architectures and planning systems such as MMOPS (Mars Mission On-Board Planner and Scheduler) [163, 164] have been already conducted. However, MMOPS is very conservative: autonomy on-board is reduced to timeline validation, control and repair, no opportunistic science is considered and most of the heavy work should be accomplished on-ground.

Mars Sample Return (MSR)

Future missions such as MSR are key to understand the requirements and research lines of future planning systems.

The MSR is an international mission aiming to return 500 grams of Mars samples to the Earth in order to be analysed with instrumentation only possible in Earth-based laboratories. The current MSR architecture is based on a campaign of three missions plus a facility for Mars Sample handing on Earth (see Figure 2.2):

- A Sample Caching Rover (SCR) mission (2018 or later), which acquires and places the samples inside the cache for later pickup.
- A MSR Lander mission (2024) with a Sample Fetching Rover (SFR) which searches and retrieves the cache, places it inside the orbiting sample (OS) container and launches it into a low Mars orbit.

- a MSR Orbiter Mission (2022), which searches and captures the OS, places it in a bio-container and brings it back to Earth.
- A Mars Sample receiving facility which retrieves the bio-container for delivery to the Sample Receiving Facilities for samples unpacking, before distributing them to the science community.

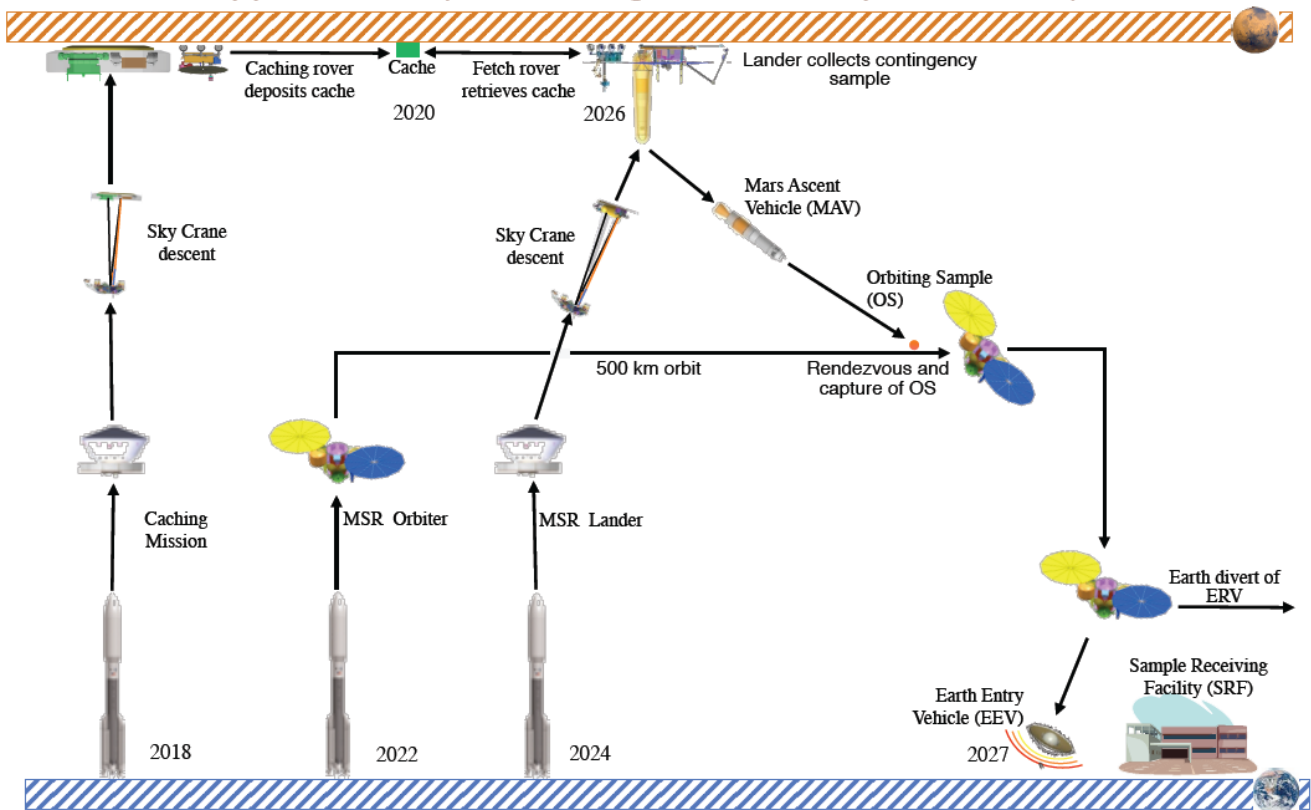


Figure 2.2.: Mars Sample Return mission (Courtesy of NASA).

Both, the SCR and SFR will require unprecedented levels of autonomy due to time constraints between the different missions. The SCR will have to autonomously drive distances in the range of 10s of kilometres to find the samples in a time frame delimited by the arrival of the SFR. This one will have to navigate to the location of the sample cache and retrieve it to the Mars Ascend Vehicle (MAV) also within a restrictive timeline. In order to mitigate a possible SCR malfunction, the SFR will itself acquire soil samples from the surface/underground while travelling along its landing site and deliver them to the MAV [6, 105].

On-Ground Domains

Robots have just recently started to move from highly structured domains like factories to the open world. Some examples are unmanned vehicles (ground, aerial and underwater), for surveillance/military purposes, rescue robots, robots playing different sports (football, ping-pong, . . .), autonomous drilling in mines, robotic vacuum cleaners for domestic use, etc. Following the schema of the previous space domains, the need of autonomy on-ground is justified and then the most relevant scenarios are presented.

Reasons for Autonomy - Lessons Learned from Fukushima Disaster

On March 11, 2011, a 9.0 magnitude earthquake and following tsunami hit eastern Japan. At the Fukushima Daiichi Nuclear Power Plant, at least three nuclear reactors suffered explosions due to hydrogen gas that had built up within their outer containment buildings after cooling system failure resulting from the loss of electrical power. Due to the high levels of radioactivity, mobile rescue robots were selected for surveillance missions.

Less than a week after the disaster, iRobot sent two 510 PackBot, spare parts and engineers to the disaster area. Since then, several robots such as iRobot 710 Warrior, Quince 1&2 or Monirobo have been used for inspection in order to obtain information about the environment. In the future, robots will be expected to conduct sampling and decontamination activities.

All these robots are tele-operated, imposing a high workload. To decrease it, Quince team offered a shared autonomy system similar to those used in the Rescue League [113] but it was rejected by TEPCO operators as they did not trust the system. Tele-operation imposes several time-consuming restrictions [82]:

- **Training:** The company in charge of the nuclear plant did not want external personnel to directly take part in the missions. It took around one week after the arrival of the two 510 PackBot to train internal personnel and six weeks for the Quince, as it was assigned more complex activities.
- **Limited connection:** Wireless communications were very limited for two reasons: (1) Limited connectivity due to the reactors' thick concrete walls and metallic components (2) The noise created by the radiation. Tethers were used instead, trading off range and mobility. With a fiber-optic tethers the range is limited to a maximum of two kilometres, forcing the operators to work nearby dangerous areas.
- **Limited dexterity:** Operators had to manipulate robot controls with protective clothing such as gloves and masks, which highly limited their dexterity.
- **Situation awareness:** Due to radiation, most sensors such as GPS have noise. The images provided by cameras flicked and had a very limited field of view.
- **Complex operations:** Operators had to face complex manoeuvres, making progress slow and putting the robots in danger. Two relevant examples are: (1) At the start and end of climbing stairs, the arm had to be moved to adjust the robot's center of gravity or it could flip over (2) The controller pad was the same as those used for video games, requiring to switch between the driving mode and the arm-control mode continuously, on top of which, this had to be done in the middle of unstable tasks, such as climbing stairs.

These restrictions are likely to be present in many other disaster scenarios apart from Fukushima and could be addressed, at least partially, by endowing the robots with appropriate levels of autonomy.

RoboCup Rescue League

Its objective is to develop and demonstrate advanced robotic capabilities for emergency responders using annual competitions and workshops to evaluate and disseminate best-in-class robotic solutions.

The league hosts annual competitions to (1) Increase awareness of the challenges involved in deploying robots for emergency response applications; (2) Provide objective performance evaluations of mobile robots operating in complex yet repeatable environments; and (3) Promote collaboration between researchers.

The scenario is the following: A building has partially collapsed due to an earthquake. Fearing secondary collapses from aftershocks, teams of robots are used to search the interior of the building for victims. The mission for the robots and their operators is to build a map of the area, find victims, determine their situation, state, and location, and then report back their findings in a map of the building with associated victim data. The section near the building entrance appears relatively intact while the interior of the structure exhibits increasing degrees of collapse. The robots are considered expendable in case of difficulty.

Specific robotic capabilities encouraged in the competition include the following: Negotiate compromised and collapsed structures; Locate victims and ascertain their conditions; Produce maps of the environment; Establish communications with victims; Deliver fluids, nourishment, medicines; Emplace sensors to identify/monitor hazards; Mark or identify best paths to victims.

Autonomy remains so far in the background. At present, all Rescue League teams rely on semi-autonomous systems with an operator in the loop. Once the robot starts a mission, it has pre-programmed behaviours to achieve the goals, which might be considered equivalent to an E3 level of autonomy. If required, the operator can manually assign specific tasks to the robot, from high level such as find victim to low level such as manual driving.

However, different scenarios such as oil spills, nuclear accidents or natural disasters requiring humanitarian help where time is critical might require higher levels of autonomy.

Darpa Robotic Challenge (DRC)

The Fukushima event inspired DARPA to create the DRC competition. Similarly to the Rescue League, DRC aims to speed up the development of robots used in response to natural and man-made disasters.

The competition was divided in eight tasks:

- **Drive and Exit Utility Vehicle:** The robot must be able to safely drive a vehicle despite occasional communications disruptions. Just getting out of the driver's seat poses significant strength and dexterity challenges.
- **Walk Across Rough Terrain:** The robots must maintain their balance and identify safe routes for placement of limbs.
- **Remove Debris from Doorway:** Robots must demonstrate a wide range of motion, in addition to balance and strength, to clear the path forward.
- **Open Series of Doors:** Moving the doors in an arc challenges the robots perception and dexterity. The robots must figure out how to align and move themselves as they open each door.
- **Climb Industrial Ladder:** To avoid falls, the robots must safely navigate the ladder and maintain their balance as they climb. Strength is key in this task.
- **Cut Through Wall:** Using power tools tests the robots' strength, dexterity and ability to perceive their environment. The robots must also simultaneously apply rigid force to hold a tool, yet demonstrate the flexibility to smoothly manipulate it.
- **Carry and Connect Fire Hose:** The robots must identify the standpipe and then transport a bulky, non-rigid item (the fire hose) to it. The robots must then have sufficient dexterity and strength to attach the hose to a standpipe and open the spigot.

- **Locate and Close Leaking Valves:** The robots must identify the valves, determine which ones are open and have sufficient range of motion to turn the valve wheels in an arc to close them.

These tasks require the following key capabilities: Mobility and dexterity to manoeuvre in the degraded environments typical of disaster zones; Ability to manipulate and use a diverse assortment of tools designed for humans; Ability to be operated by humans who have had little to no robotics training; Partial autonomy in task-level decision-making based on operator commands and sensor inputs.

In this case, the robots are mostly operated through semantic commands, which represents a level of autonomy even lower than in the Rescue League due to the higher complexity of the tasks and hardware. The operators in a control station command and supervise the robot as it accomplishes the different tasks in a semi-autonomous way. First the robot generates a 3D map consisting on a cloud of points and the operators later match the points to specific objects on a database. Following, the operator indicates to the robot the next action, which could represent complex or simple behaviours. For a complex behaviour, the operator can select an object in the environment and through commands like walk and grasp, the robot can autonomously generate a footstep plan to get near the object and estimate the position of the hand needed to grasp it. Or for a simple behaviour, the operator can select an element of the robot such as the hand and move it in the virtual environment to the target point. Once the action is received, the robot calculates the trajectory and performs it.

Comparing the Mars and Rescue Scenarios

Disaster environments and planetary exploration present several similarities and differences illustrated in Table 2.1. In consequence, both test cases complement each other, helping to better understand the possibilities and requirements of automated planning and execution for robots.

Env.	Dynamic	Observ.	Struct.	Reconf.	Ops.	Autonomy	Recovery	Cost	CPU
Mars	✓	✗	✗	✗	✗	High	Impossible	\$2500M	Low
Rescue	✓	Maybe	✗	✓	Maybe	Low-High	Possible	\$100K	High

Table 2.1.: Relevant features of “real-world” robotic scenarios: (3) Observable: Indicates whether the environment is observable; (4) Structured; (5) Reconfigurable: Possibility of reconfiguring the robots for other missions; (6) Direct operations: Indicate whether the robot can be directly operated or not; (8) Recovery: Possibility of recovery under critical failure.

With respect to the environments, both of them are very similar. Only in terms of observability might the former have an advantage thanks to the high availability of solutions on Earth such as cameras, human inspection, aerial vehicles, etc, which obviously is not accessible in space.

Regarding the robot, the criticality is quite different mainly because major malfunctions in a Mars rover use to represent the end of the mission, while rescue robots can be fixed or replaced. A related factor is the cost of each robot/mission which is one of the main limitations to send robots to other planets.

The type of missions is also very different: in the case of planetary rovers, they are designed to conduct very specific tasks, making their adaptation to different uses unlikely. On the other hand, rescue robots need to be highly reconfigurable.

With respect to the level of autonomy, rescue scenarios offer different options depending on the type of environment, from tele-operation to highly automated. On Mars, due to several reasons including

signal delay, limited communications, etc. operations use to be highly automated while tele-operation is completely discarded.

2.3 Planning Formalisms

The first step towards automated problem solving consists on providing the theoretical foundations in the form of a formalism used to explain fundamental concepts about planning, formally define domains and problems or prove theorems.

There are multiple forms of planning, each specially suited for some specific type of problem. The following is a general classification [80]:

- Classical Planning:
 - State-Space Planning: The search space is a (maybe infinite) tree or graph representing all the possible states of the problem where each node is a fully defined state and each arc represents the transition from one state to another. From the initial state, a number of children are generated, one for each applicable action and one of them becomes the current state from which search continues until a solution is met. It will be further explained in Section 2.3.1.
 - Plan-Space Planning: The search space can be represented as a tree or graph where the nodes are partially defined plans and arcs represent plan refinements intended to complete a partial plan. Planning starts from an initial node corresponding to an empty plan, searching for a solution by means of two operations: *choosing* an action (as in state-space) and *ordering* it. A plan is defined as a set of planning operators together with ordering constraints. Even though it is included as a classical technique, it represents the most common approach for timeline planners (TLP), which are of special interest in the thesis.
- Neoclassical Planning:
 - Graphplan Planning: The search space is a directed layered graph, where each layer i is divided in two nodes: one with actions (A_i) and one with propositions (P_i). The initial state P_0 contains all initial propositions. Search starts from P_0 to generate A_1 , the set of actions (ground operators) whose preconditions are satisfied in P_0 . P_1 is the union of P_0 and the set of positive effects of actions in A_1 . This process continues from one layer to the next. A plan is a sequence of sets of actions $\Phi = \langle \pi_1, \pi_2, \dots, \pi_k \rangle$, where π_i is a subset of independent actions of A_i that can be applied in any order to P_{i-1} and lead to a state which is a subset of P_i .
 - Planning as Satisfiability: The idea is to formulate a planning problem as a propositional formula. A satisfiability decision procedure determines whether the formula is satisfiable by assigning truth values to the propositional variables. Finally, a plan is extracted from the assignments determined by the satisfiability decision procedure.
 - Constraint Satisfaction: Given a set of variables, their domains and a set of constraints on the values that the variables may take, the problem is to find a value for each variable within its domain such that the set of values for all the variables meet all the constraints.
- New Planning Systems:
 - Hierarchical Task Network Planning: It is similar to classical planning as each state of the world is represented by a set of atoms and each action corresponds to a deterministic

state-transition, but it is different in the following sense: Planning proceeds by *reducing* non-primitive tasks recursively into smaller and smaller subtasks, until the point where all of them are primitive.

- Temporal Planning: In this case, a state represents the value assumed for each variable in a specific instant of time. An action is not a single state-transition but a collection of local change and persistence conditions that are spread out in time but are focused on just a few state variables or propositions.
- Planning with Uncertainty:
 - Planning based on Markov Decision Processes (MDPs): Represents the planning problem as an optimization problem. The domain is modelled like a non-deterministic state-transition system that assigns probabilities to state-transitions. Goals are utility functions used to express preferences on the entire execution path of a plan rather than just desired final states.
 - Planning based on Model Checking: Its key idea is to solve planning problems model-theoretically. The domain is modelled like a non-deterministic state-transition system where an action may lead from one state to many different states. Goals are temporal logic formulas.

Most modern classical planners follow an early commitment approach based on forward-chaining heuristic search in the state-space. In particular in satisficing planning (where planners are not required to find optimal solutions but rather solutions with reasonable quality quickly), state-space heuristic search has outperformed other approaches by a large margin in the past decade. Since 2000, for example, all winners of the classical satisficing track in the roughly biennial IPC have followed this paradigm (e.g [91, 86, 38, 127]). Recent developments, however, suggest that this picture may change (e.g. [130] for a highly competitive planner based on satisfiability). In addition, state-space heuristic search has also become the state-of-the-art technique for optimal planning (see [97, 87]).

On the other hand, applied planners follow a least-commitment approach based on Partial Order Planning (POP) for the plan space and require a combination of technologies in order to find solutions to problems that are theoretically intractable. For instance, ASPEN, EUROPA and APSI (see Section 2.4.4) are based on a combination of a heuristic iterative repair unfold for subgoaling, a scheduler to handle resources, a temporal solver to handle time and a Constraint Satisfaction Problem (CSP) to solve parametric constraints.

The most influential approaches for this thesis are forward-chaining heuristic search in the state-space, Timeline and HTN planning, which have been merged with other techniques such as partial and parallel planning. The resulting planner, named QuijoteExpress [174, 167, 175], is the first timeline planner based on state-space to the best of our knowledge.

Next, all the cited formalisms relevant for QuijoteExpress are reviewed in depth.

2.3.1 State-Space

STRIPS (Stanford Research Institute Problem Solver) [68] is by far the most widely used state-space formal language to date due to its simplicity and reasonable expressiveness. Many languages are based on STRIPS such as the **PDDL** [104] family or *SAS⁺* [8]. The following sections introduce a general overview of its main concepts, based on [81] which is very close in many aspects to APSI.

State Variable

Definition 3 (State-variable) A k -ary state-variable $x(v_1, \dots, v_k)$ is represented by a symbol x and each v_i is either a constant or a variable with an associated domain D^{v_i} . A state-variable is **ground** if every v_i is a constant, or **unground** otherwise.

Definition 4 (State) A state s is an assignment of values to all the variables¹, formally: $s = \{(x = c) | x \in X\}$ where $c \in D_x$, that is, a list of values in s of all the ground state variables.

Example 1 In the example of the two collaborative Mars rovers, the location of the primary robot at a given state s can be expressed as follows: $rloc(r_{primary}) = loc1$
where $rloc$ is a state variable and $r_{primary}$ is a variable used to designate one of the robots².

Problems use to have facts that are invariant along all the states. They are represented as **relations** which share the same structure as state variables, but in this case the list of variables is replaced by a list of subdomains. The list of relations of a domain is represented as R .

Example 2 The following constraint states that a satellite will provide a communication link to the primary rover in the location p_{dest} :

$$visible(sat_1, p_{dest})$$

These facts cannot be changed by the planner, therefore do not need to be stated in each state.

Operators and Actions

Definition 5 (Operator (o)) It is a triple $o = (name(o), precondition(o), effects(o))$ where $name(o)$ is a unique name associated to a list of parameters, $precond(o)$ is a set of expressions on state variables and $effects(o)$ is a set of assignments to state variables.

Example 3 Continuing with the Mars rovers, a formal description of an operator to move one of the robots could be: $going_to(r_{primary}, p_{orig}, p_{dest})$

$$precond: rloc(r_{primary}) = p_{orig}$$

$$effects: rloc(r_{primary}) \leftarrow p_{dest}$$

The conditions in $precond(a)$ refer to the values of state variables in a state s , whereas the updates in $effects(a)$ refer to the values in the state $\gamma(s, a)$ (see Definition 11).

Definition 6 (Action (a)) It is a grounded operator o that satisfies all the preconditions in $precond(o)$.

An action is **applicable** in a state s iff the values of the state variables in s satisfy the conditions in $precond(a)$, formally: $\forall (x = c) \in precondition(a) | (x = c) \in s$. In that case, the state $\gamma(s, a)$ is produced by updating the values of the state variables according to $effects(a)$.

Definition 7 (Goal) State specified by defining the values of one or more grounded state variables.

Example 4 The following goal indicates that the robot $r_{primary}$ must be in position p_{dest} :

$$at(r_{primary}) = p_{dest}$$

¹ In plan-space a state is a partial assignment.

² Since all state variables depend on the current state, $x(v_1, \dots, v_k)$ refers implicitly to the value of x in the current state s .

Search Space

Definition 8 (Search space) *The set of all reachable states from the initial state by any sequence of actions, which might be infinite. Depending on the properties of the domain, the topology of the search space can be a tree, in case there is just one way to get from one state to another, or a graph otherwise.*

Each node in the search space contains a fully defined state of the world while arcs correspond to state transitions. The initial state is known as the **root** node while a **leaf node** is the one with no children in the search tree (see below).

The search space is typically implemented with two lists.

Definition 9 (Open-list) *Set of all leaf nodes available for expansion at any given point, also known as **frontier** because it separates the explored region of the graph from the unexplored.*

Definition 10 (Closed-list (optional)) *Set of nodes already visited. It is used in case the search space is a graph to avoid exploring more than once those nodes in loopy paths.*

Domains and Problems

Definition 11 (Domain (Σ)) *It is modelled as $\Sigma = (S, A, \gamma)$ such that:*

- S : *The search space.*
- A : *Set of actions that meet the relations in R .*
- $\gamma(s, a)$: **State-transition function** such that $\gamma(s, a) = \{(x = c) | x \in X\}$, where c is specified by an assignment $x \leftarrow c \in \text{effect}(a)$ or $\{(x = c)\} \in s$.

It is important to remark the difference between states and nodes. A state represents a configuration of the world given by a partial assignment to the variables while a node is a bookkeeping data structure that contains, besides the state, relevant information for the search algorithm such as the parent-node (to extract a path), cost (to evaluate a solution), h -value (to evaluate distance to the goal), etc.

Given a state-transition with two states $s, s' \in S$ and one action $a \in A$ such that $s' \in \gamma(s, a)$, there is an arc from s to s' labelled with a .

Definition 12 (Problem (P)) *A **problem** is a triple $P = (\Sigma, s_0, g)$ where s_0 is the initial state and g a set of expressions on the state variables representing a goal state that might be partially defined, that is, the value for some state variables might not be specified.*

The Problem of Planning

A **plan** for a problem P is a sequence of actions $\Pi = \langle a_1, \dots, a_k \rangle$ such that a_1 is applicable to s_0 , a_2 is applicable in $\gamma(s_0, a_1)$ and so forth, and the conditions in the goal g are met in $\gamma(s_{k-1}, a_k)$. The process of looking for a plan is called **search**.

Search can proceed from the initial state towards the goal (forward-search) or the other way around (backward-search). The former is generally accepted nowadays to be faster because the initial state is

fully defined while the goal state is not, which helps to decrease the branching factor. Regardless of the search direction, a plan can be extracted following the path from the initial state to the goal state.

When search starts, the first step is to test whether the current state is a goal. If it is not, then the node is **expanded**, that is, apply each applicable action to the current state, thereby generating a new set of states (one for each applicable action). Next, the current state must be moved to one of the new generated states and the process repeats until the **end condition** is met, process that grows a **search tree** that will cover the search space (see Figure 2.3). End conditions can be defined in terms of some limit such as maximum search time, memory used, number of nodes expanded, etc. or in terms of the search type where the condition is met when one random goal is reached (satisficing planning), the optimal goal is reached (optimal) or all goals are reached (complete).

Most search algorithms follow this process; they vary primarily on how they choose which state to expand next, the so-called **search strategy**.

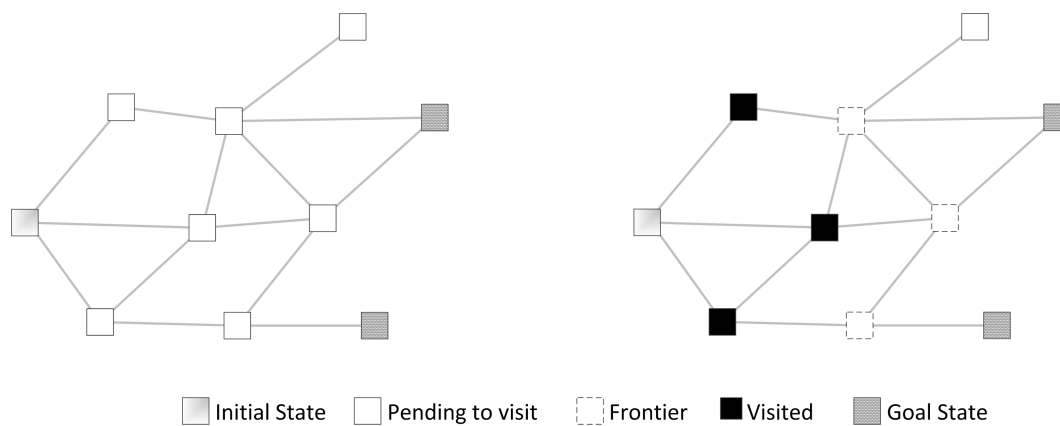


Figure 2.3.: Search tree generated during planning covering the graph search space.

PDDL

The Planning Domain Definition Language (PDDL) [104, 69, 59] is a “de facto” standard in classical planning and the official language for the International Planning Competition (IPC) since 1998. Several versions have progressively increased its expressive power, usually ahead of the capabilities of its contemporary planners.

PDDL is a first-order logic, action-based language inspired by STRIPS with a finite number of predicate symbols, constants and variables. States are sets of ground atoms while actions are defined as (grounded) planning operators with preconditions and effects, expressed as logical propositions, and can be parametrised by means of variables (terms). The version 2.1 added several extensions such as numeric expressions, metrics, spontaneous events and, more importantly, *durative* actions used for temporal planning.

Like in many other languages, modelling with PDDL is divided in two parts: domain and problem. Most modern planners do not use PDDL representation internally, but translate PDDL tasks into simpler formalisms and ground all operators before planning [129].

2.3.2 Hierarchical Task Networks (HTN)

Hierarchical Task Networks (HTN) [134, 144, 165, 96, 64, 62, 157] is one of the most used planning techniques for real world applications [159], in part because it allows to effectively encode knowledge into domain-independent planners. Some relevant HTN planner examples are SIPE-2 and SHOP2 (see Sections 2.4.3 and 2.4.3), Nonlin [144], one of the first HTN planning systems or O-Plan [145].

HTN planning uses actions and states of the world that are similar to those used in STRIPS-style planning, i.e., operators consisting of three lists of atoms: a precondition list, an add list, and a delete list. However, in HTN planning the objective is not to achieve a final goal as for STRIPS-style, but to perform instead some set of tasks. The definitions of term, literal, atom, action and transition function are the same as in classical planning.

The input to the planning system includes a set of operators similar to those of classical planning and a set of *methods* which describe how to *decompose non-primitive* (or complex) tasks into subtasks. HTN methods provide a convenient way to write problem-solving “recipes” that correspond to how a human expert might think about solving a planning problem. Methods make it much easier to solve the planning problem because they generate only plans that are solutions to the problem. Non-primitive tasks cannot be directly executed as they could be achieved in different ways. For example, if a robot has to go from point a to b, it could do it with auto navigation or teleoperated. Planning proceeds by decomposing *non-primitive tasks* recursively into smaller and smaller subtasks, until all are *primitive tasks*.

HTN is based on a first-order language with some extensions, formally: $L = \langle V, C, P, F, T, N \rangle$ where V is a set of variable symbols, C constant symbols, P predicate symbols, F primitive-task symbols, T compound-task symbols, and N symbols used for labelling tasks.

Definition 13 (Task) *A task is a syntactic construct $\tau(x_1, \dots, x_n)$ where $\tau \in T$ and x_1, \dots, x_n are terms. There are two type of tasks: primitive, called operators and nonprimitive called methods. The task is grounded if all of the terms are grounded; otherwise, it is ungrounded.*

Definition 14 (Method) *A method is a tuple $m = (\tau, \omega)$ where τ is a non-primitive task and ω is a task network. It states that one way to accomplish the task τ is to achieve all the tasks in ω without violating ω 's constraints.*

A task (operator or method) τ is *applicable* in a state s if $precond^+(\tau) \subseteq s$ and $precond^-(\tau) \cap s = \emptyset$. A method $m = (\tau, \omega)$ is *relevant* for a task τ_i if $\tau = \tau_i$. An action $a = (name(a), precond(a), effects(a))$ accomplishes a ground primitive task τ in the state s if $name(a) = \tau$ and τ is applicable to s .

Definition 15 (Constraints) *A constraint is a boolean formula. Being $v, v' \in V$, l a literal, $c \in C$ and $n, n' \in N$, the type of constraints in HTN are:*

- *Variable binding constraints such as $v = v'$ or $v = c$.*
- *Ordering constraints such as $n \prec n'$ means that the task labelled with n precedes the one labelled with n' .*
- *State constraints such as n, l , l, n or n, l, n' , meaning that l must be true after n , before n or between n and n' respectively.*

Definition 16 (Task network) *A task network is a pair $\omega = (N, C)$ in which N is a set of task nodes and C the set constraints. Each node $n \in N$ contains a task τ_n , represented as $(n : \tau_n)$. If all tasks τ_n*

are grounded, then ω is grounded. Each edge $c \in C$ contains a constraint. If all nodes are ordered, then ω is totally ordered. If all nodes are primitive, then ω is primitive.

Definition 17 (State) It is a list of ground atoms. The atoms appearing in that list are said to be true in that state and those that do not appear are false in that state.

Definition 18 (Domain) A domain D is a pair $D = \langle O, M \rangle$ where O is a list of operators and M a list of methods.

Definition 19 (Problem) A problem instance P is a triple $\langle \omega, I, D \rangle$ where ω the target task network that requires planning, I is the initial state and D is a planning domain.

P is primitive if the task network d contains only primitive tasks. P is regular if all the task networks in the methods and d contain at most one non-primitive task, and that non-primitive task is ordered to occur as either the first or the last task. P is propositional if no variables are allowed. P is totally ordered if all the tasks in any task network are totally ordered.

A detailed description of a planning domain and problem for both the planetary rover and rescue robot scenarios are given in Chapter 6.

Definition 20 (Decomposition) Let $\omega = (N, C)$ be a non-primitive task network that contains a (non-primitive) node $n_i : \tau_{n_i}$. Let $m = (\tau_m, \omega_m)$, where $\omega_m = (N_m, C_m)$ be a relevant method for τ_{n_i} and θ the most general unifier of τ_{n_i} and τ_m . The result of decomposing n_i by ω_m in ω under θ is obtained as follows:

$$\delta(\omega, n_i, m, \theta) = \omega' | \omega' = (N', C') \quad (2.1)$$

where:

$$N' = N - n_i \cup \theta(N_m)$$

C' is obtained from C by making the following changes:

- replace $(n_i \prec n_j)$ with $last(N_m) \prec first(N_{n_j})$, that is, all tasks in the decomposition of n_i must precede all tasks in the decomposition of n_j .
- replace (l, n_i) with $(l, first(N_m))$.
- replace (n_i, l) with $(last(N_m), l)$.
- replace (n_i, l, n_j) with $(last(N_m), l, first(N_{n_j}))$.
- replace n_i by N_m in every $first[]$ or $last[]$ expression where it appears.

Definition 21 (Solution) Let $P = \langle \omega, I, D \rangle$ a planning problem. The cases in which a plan $\pi = \langle o_1, \dots, o_m \rangle$ is a solution for P are:

- ω is empty: Then, the empty plan $\pi = \langle \emptyset \rangle$ is a solution.
- The first task $(n : \tau_n)$ of ω is a primitive task: π is a solution for P if τ_n is applicable to s_0 and the plan $\pi = \langle o_2, \dots, o_m \rangle$ is a solution for $P' = (\omega - n, \gamma(s_0, a_1), D)$, where P' is the planning problem obtained from executing the first task of π (τ_n) and removing its corresponding node (n) from the task network.

- The first task ($n : \tau_n$) of ω is a nonprimitive task: In case there is a method m relevant for τ_n and applicable in s_0 . Then π is a solution for P if there is a task network $\omega' = \delta(\omega, n, m, \theta)$ such that there is a solution for $P' = (\omega', s_0, D)$.

The constraint formulas are evaluated as follows:

- $(c_i = c_j)$: true, if c_i, c_j are the same constant symbols.
- $first[n_i, n_j, \dots]$: returns the first node.
- $last[n_i, n_j, \dots]$: returns the last node.
- $c = (n_i \prec n_j)$: true if $c \in C$.
- l, n_i : n_i holds in s_j , then true if l holds in s_{j-1} .
- n_i, l : n_i holds in s_j , then true if l holds in s_j .
- n_i, l, n_j : n_i holds in s'_j , n_j holds in s''_j , then true if l holds in all s_e such $j' \leq e < j''$.

If π is a solution for (ω, s_0, D) , then for each task node $n \in N$ there is a decomposition tree whose leaf nodes are actions of π . The decomposition tree is composed of AND-branches. If n is primitive the decomposition tree consists just of n . Otherwise, if n decomposes into a task network $\delta(\omega, n, m, \theta)$, then the root of the decomposition tree is n and its subtrees are the decomposition trees for the nodes of $\delta(\omega, n, m, \theta)$. Detailed information about the solution structure for the Mars rover scenario is given in Chapter 6.

Different properties of HTN useful for HTLN such as expressiveness, soundness, completeness or efficiency will be analysed in Chapter 3.

2.3.3 Timeline Planning

Representing time in applied planning systems is important because actions and relations occur over timespans [53, 3, 153, 81, 39]. The STRIPS-style model of an action based on preconditions which happen *before* the action, and effects happening *after* it must be greatly extended to represent the different type of temporal relations between two or more actions/states. Several planners systems such as ASPEN, EUROPA, APSI or IxTET are good examples of temporal planning, more specifically timeline-based planning³.

A TLP problem is defined as a set of constraints (conjunctive or disjunctive) over a set of temporal elements (time points or intervals) assigned to some actions. It can be resolved applying generic CSP techniques, even though specific algorithms to deal with resources management have been developed. The solutions are represented as timelines or sequence of states defining the behaviour of the system. The main concepts are next introduced in detail.

Definition 22 (Temporal reference, time point, time interval) *A temporal reference is an instant or interval. An instant is a variable ranging over the set \mathbb{R} of real numbers. An interval is a pair (x, y) of instants, such that $x < y$. Some examples of temporal relations are before, overlaps or starts. They can be expressed as CSP problems using two different formalisms: time-point algebra and interval algebra.*

³ In this thesis the term *Timeline-based planning* (TLP) is used instead of *Constraint-Based Interval planning* (CBI) [71] because the former is more widely accepted

Definition 23 (Point algebra (PA) [153]) PA defines how to relate in time a set of instants with qualitative constraints without necessarily ordering them. Given a set of primitive relation symbols $P = \{<, =, >\}$, the possible qualitative constraints are:

$$R = \{\emptyset, \{<\}, \{=\}, \{>\}, \{<, =\}, \{>, =\}, \{<, >\}, P\} \quad (2.2)$$

where \emptyset is the empty constraint and P the universal. A constraint is represented as $[t_i r t_j]$, where t_i, t_j are time instants and r a relation symbol. The symmetrical constraint of r is r' , obtained by replacing $<$ by $>$ and the opposite.

A binary constraint network for the point algebra is defined as a directed graph (X, C) where $X = \{t_1, t_2, \dots, t_n\}$ is a set of instant variables and each arc in C is a constraint. In contrast to standard CSPs, variable domains in PA are infinite. A tuple of real numbers (v_1, \dots, v_n) is a solution of (X, C) iff the n values $t_i = v_i$ satisfy all the constraints of C .

Definition 24 (Interval algebra (IA) [3]) IA is similar to PA except that it deals with intervals instead of points. Two intervals x and y whose end points are precisely set in \mathbb{R} can be related qualitatively in only thirteen possible ways composed of the seven represented in Table 2.2 plus their symmetrical relations.



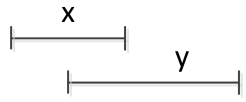
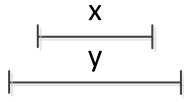
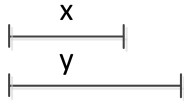
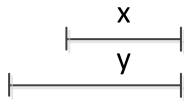
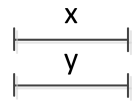
Relation	Meaning
before	
meets	
overlaps	
during	
starts	
finishes	
equal	

Table 2.2.: Allen's interval relations.

Let P be the set of thirteen primitive relations, then R can be any combination of them plus the symbols \emptyset and P itself. Each constraint $r \in R$ is a constraint interpreted as the disjunction of these primitives. For example, $x \{before, meets\} y$ denotes $(x \text{ before } y) \vee (x \text{ meets } y)$.

The definition of binary constraint network for IA is equivalent to this of PA replacing instant variables by interval variables. A tuple of pairs of real numbers $((v_1^-, v_1^+) \dots (v_n^-, v_n^+))$, with $v_i^- \leq v_i^+$, is a

solution of (X, C) iff the n intervals v_i^-, v_i^+ satisfy all the constraints of C . The IA network (X, C) is consistent when a solution exists.

Definition 25 (Simple temporal constraint problem (STP) [53]) Let $X = \{t_1, t_2, \dots, t_n\}$ be a set of time-points, each with domain in \mathbb{R} . The following two constraints can be applied over X :

- *Unary*: $a_i \leq t_i \leq b_i$.
- *Binary*: $a_{ij} \leq t_j - t_i \leq b_{ij}$.

with $a_i, b_i, a_{ij}, b_{ij} \in \mathbb{R}$. Each unary constraint affecting t_i can be written as binary by taking a reference point t_0 as follows: $a_i \leq t_i - t_0 \leq b_i$. A simple temporal constraint problem (STP) is a pair (X, C) where each element in C is an interval r_{ij} that constrains the relative distance of a pair of instants (t_i, t_j) . A STP (X, C) is consistent if there is a solution that satisfies all the constraints. It is minimal if every point in an interval r_{ij} belongs to some solution.

Definition 26 (Temporal constraint satisfaction problem (TCSP) [53]) In the general problem, called TCSP, disjunctions of constraints are allowed on the distances between pairs of instants. In a TCSP network, the constraint between a pair (t_i, t_j) corresponds to the disjunction $(a_l \leq t_j - t_i \leq b_l) \vee \dots \vee (a_m \leq t_j - t_i \leq b_m)$, denoted as $r = \{[a_l, b_l], \dots, [a_m, b_m]\}$. Consistency and minimality properties are equivalent to those of STPs.

Most of the temporal planners cited in this chapter share a common timeline-based approach to mission planning. In this approach, the overall system being planned (the spacecraft and relevant environment elements) is represented by a set of timelines. In spite of recent efforts to formally define timelines [39, 70], there is no consensus on what a timeline exactly is. A general definition could be the following.

Definition 27 (Timeline [70]) Complete history of value changes of a variable over the specified planning horizon.

There are several types of timelines. In [70] they are classified according to the following criteria:

- *Variable type*: A timeline can be a state variable that usually models a given subsystem (e.g. camera, robotic arm, etc.) or a resource (e.g. power, memory, etc.).
- *Value constraints type*:
 - *Value*: Constrain a temporal assertion to a single value.
 - *Simple*: Constrain the value and time.
 - *Complex*: Exclude a subset of values on a timeline over a specified interval.
- *Temporal constraints type*:
 - For state variables:
 - * *Fixed*: all start and end times of temporal assertions are fixed, and all intervals are totally ordered.
 - * *Total Order*: Flexible start and end times of temporal assertions are allowed (with the implication that interval durations are flexible), but all intervals on the Timeline are totally ordered.

- * Partial Order: All start and end times of temporal assertions are fixed, but intervals may be partially ordered.
 - * Flexible Partial Order: Durations of temporal assertions are fixed, but arbitrary temporal constraints between events are otherwise supported.
 - * STN: Arbitrary temporal constraints may be imposed on the events.
 - * DTN: Disjunctive temporal network.
- For resources:
- * Fixed: All event times are fixed (but events may be simultaneous).
 - * Total Order: All event times are totally ordered but event times may be flexible.
 - * STN: Arbitrary temporal constraints may be imposed on the events.
 - * DTN: Disjunctive temporal networks.

According to this classification, there are eighteen possible specializations for state variable timelines and twelve for resources, all implying different time and space $O(n)$ complexities.

A temporal planner typically operates on a temporal database that maintains temporal references and provide functions for querying, updating, and maintaining its consistency. The planner asserts relations among these temporal references. Once an activity plan is generated, the timelines are extracted and used to verify the safety and validity of the plan.

Timeline-based planners have a number of modules, each solving a specific aspect of the problem:

- Parameter Constraint Engine: Reasoning about parameter dependencies.
- Temporal Engine: Reasoning about the temporal dependencies.
- Search Engine: In charge of adding supporting decisions to justify the new decisions added to the problem, process also known as subgoaling [101, 9, 74].
- Timeline Completer: Responsible for representing the chronology of values of a state variable, along with the temporal constraints that arise from the current chronology.

2.3.4 Planning under Uncertainty

The assumptions presented in Section 2.2.2 required for the two reference scenarios prevent the use of the classical definition of solution, as it will frequently happen that the required information to generate a detailed plan is not yet available at planning time. It might be not even desirable to generate a fully defined plan in such conditions, as the planner might be forced to be very conservative. However, a valid plan is required in order to start the execution. Two techniques have been widely used for planning under uncertainty: Markov Decision Processes [27] and Model Checking (MC) [47, 13].

Markov Decision Process

MDPs deal with nondeterminism, probabilities, partial observability and extended goals. There are two main categories: MDPs with full observability and MDPs with partial observability, also known as POMDPs which is closer to the properties of the Mars rover and rescue robot domains.

A stochastic system is a non-deterministic state-transition system with a probability distribution on each state-transition. It is a tuple $\Sigma = (S, A, O, T, \Omega, R)$ where:

- S : Finite set of states.
- A : Finite set of actions.
- O : Finite set of observations.
- $T_a(s, s')$ where $a \in A, s, s' \in S$: Probability that action a in state s will lead to state s' .
- $\Omega_a(o|s')$ where $a \in A, s' \in S$: Probability of observing o in state s' after executing action a .
- $R_a(s, s')$: Expected reward received after transition to state s' from s

At each time, the system is in some state $s \in S$. The agent takes an action $a \in A$, which causes the environment to transition to state s' with probability $T_a(s, s')$, receives a reward and the process repeats. All the information known about a state is provided by observations. It might not be possible to distinguish between two states given a set of observations. Moreover, the same state may correspond to different observations. The controller in charge of executing the plan can observe a probability distribution over states of the system called *belief state*, rather than the exact state of the system.

The problem of POMDPs is to find a “policy”, a function $\pi : B \rightarrow A$ that maps belief states to actions. Planning problems in POMDPs can be stated as optimization problems where an optimal policy π has to be generated.

Model Checking

Equally to MDPs, Model Checking (MC) deal with nondeterminism, probabilities, partial observability and extended goals.

A domain is a nondeterministic state-transition system $\Sigma = (S, A, \gamma)$, where:

- S : Finite set of states.
- A : Finite set of actions.
- $\gamma: S \times A \rightarrow 2^S$ is the state-transition function.

Non-determinism is modelled by γ : given a state s and an action a , $\gamma(s, a)$ returns a set of states. An action a is applicable in a state s if $\gamma(s, a) \neq \emptyset$. The set of actions that are applicable in state s is $A(s) = \{a : \exists s' \in \gamma(s, a)\}$.

Plans are *policies* similar to MDP policies. A policy π for a planning domain $\Sigma = (S, A, \gamma)$ is a set of pairs (s, a) such that $s \in S$ and $a \in A(s)$. For any state s there is at most one action a such that $(s, a) \in \pi$. Unlike MDP, MC policies are not necessarily defined over all states.

An execution path is a possibly infinite sequence s_1, s_2, \dots of states such that, for every state s_i either s_i is the last state of the sequence (called terminal state) or the transition $T(s_i, s_{i+1})$ holds. A state s' is reachable from another s if there is a path from s to s' .

As the execution of a plan may produce more than one possible path, the definition of a solution to a planning problem is more complicated than in classical planning. A planning problem is a triple (Σ, S_0, S_g) , where Σ is a planning domain, $S_0 \subseteq S$ a set of initial states and $S_g \subseteq S$ is a set of goal states. Let π be a policy for Σ . Then there are three kinds of solutions:

- Weak solutions are plans that may achieve the goal but are not guaranteed to do so.

- Strong solutions are plans that are guaranteed to achieve the goal in spite of nondeterminism: all the paths are finite and reach the goal.
- Strong cyclic solutions are guaranteed to reach the goal under the assumption that execution will eventually exit the loop. These solutions are such that all their partial execution paths can be extended to a finite execution path whose terminal state is a goal state.

Timeline representation is equivalent to weak solutions in MC as it is not possible to guarantee any stronger solution due to uncertainty inherent to dynamic and non-structured domains (see A2 and A3 assumptions in Section 2.2.2). To solve this situation, the system relies on replanning in case the goal is not reached.

2.3.5 Constraint Satisfaction Problems (CSP)

A constraint satisfaction problem (CSP) is defined as a tuple $P = (X, D, C)$ where:

- $X = \{x_1, \dots, x_n\}$: Set of variables.
- $D = \{D_1, \dots, D_n\}$: The set of finite domains of the variables, such that $x_i \in D_i$.
- $C = \{C_1, \dots, C_m\}$: Finite set of constraints. A constraint c_j restricts the possible values of a subset of k variables $\{x_{j1}, \dots, x_{jk}\} \subseteq X$, in other words, the subset of the Cartesian product: $c_j \subseteq D_{j1}x \dots x D_{jk}$.

A solution is a value assignment to each variable from its domain, formally a tuple $\sigma = (v_1, \dots, v_n)$ such that $v_i \in D_i$ and the values of the variables $x_i = v_i \forall i \in [1, n]$ meet all the constraints. A CSP is consistent if such solution σ exists.

An instance of a CSP can be conceptualized as a constraint graph $G = \{V, E\}$. For every variable $x_i \in X$ there is a corresponding node in V . For every set of variables connected by a constraint $c_j \in C$ there is a corresponding hyperedge in E . In the particular case in which only binary constraints (each constraint involves at most two variables) are defined the hyperedges become simple edges. A well known example of binary CSP is the Simple Temporal Network.

Given a CSP, one may be interested in addressing: (1) A resolution problem: finding a solution tuple; (2) A consistency problem: checking whether such a solution exists; (3) A filtering problem: removing some redundant values from domains or some redundant tuples from constraints; and (4) A minimal reduction problem: removing every redundant value and redundant tuple.

The consistency problem of a binary CSP over finite domains is NP-complete. Resolution and minimal reduction are NP-hard problems. Filtering is a polynomial problem. Its practical efficiency makes it desirable as an approximation to consistency checking: it provides a necessary but not a sufficient condition of consistency. If filtering reduces a domain or a constraint to an empty set, then the CSP is inconsistent, but the converse is not true. Filtering is useful in particular when a CSP is specified incrementally, as is typical for CSPs in planning, by adding at each step new variables and constraints. One needs to check whether a step is consistent before pursuing it further. This incremental checking can be approximated by filtering techniques.

Significant work has been conducted in the field of CSP planning, where different versions of arc and path-consistency algorithms have been used in several planners [103, 14, 140]. In APSI, a CSP solver is used to maintain parameter consistency. QuijoteExpress benefits from this functionality and extends it to satisfy some specific requirements from HTN to represent the fact that a compound task and its related network are actually the same thing.

2.4 Software for Autonomy

There is a wide variety of autonomous software used on ground and space domains. This section starts analysing generic search algorithms used by most automated planning systems, then it provides an overview of mission planners and generic frameworks and finally presents some expert systems used in Mars rovers.

2.4.1 Search Algorithms for Planning

To find solutions in the search space, both classical and applied planners use search algorithms. They can be classified in two main subgroups: **uninformed** and **informed**. The first ones do not have information about the search-space beyond the frontier. All they can do is to generate successors and distinguish a goal state from a non-goal state. The later ones have information that help to evaluate how far each node is from the goal.

Uninformed Search

Suppose that the search space is a graph where b is the branching factor, d the depth of the shallowest goal node and m the maximum depth of any path. Some of the more relevant uninformed algorithms are:

- **Depth-first search (DFS)**: Its strategy consists on expanding the deepest node of the frontier starting from the root node. It is complete in finite state-spaces, but not optimal as it might find a non-optimal goal first. The time complexity is $O(b^m)$, much higher than this of BFS (see below), but its memory requirements are smaller, $O(bm)$.
- **Depth-limited**: To alleviate the problem of infinite length paths in DFS, this algorithm uses a depth limit l where all nodes at l are treated as if they had no successors. However, l introduce yet another source of incompleteness when $l < d$.
- **Iterative deepening depth-first search (IDDFS)**: It searches for solutions up to a given depth typically with a DFS strategy. In case no solution is found, the maximum depth is increased and the search repeated from the beginning. The algorithm will have to expand the same node a number of times proportional to its depth. More specifically, if the solution is at depth d , the time complexity is $O(b^d)$ and the space complexity $O(bd)$. Since most of the nodes are in the bottom level, this fact is not so harmful. It is complete when the branching factor is finite and optimal when the path cost is a none decreasing function of the depth of the node.
- **Breadth-first search (BFS)**: Its strategy consists on expanding the shallowest unexpanded node. It starts expanding the root node, then all its successors, and so on. It is complete if the shallowest goal node is at finite depth. It is optimal if the step-cost increases with depth. The time complexity is $b + b^2 + \dots + b^d = O(b^d)$, the same as the space complexity, equivalent to the number of nodes stored in the open list, which dominates the number of nodes in the closed list, equals to $O(b^{d-1})$.
- **Uniform-cost search**: In case actions have different costs, BFS is not optimal. Uniform-cost is a variant that expands the node with the lowest cost $g(n)$ (also referred to as g -value) of the open list. If there is an infinite sequence of zero-cost actions, the algorithm will get stuck. Therefore, it

is complete only if every action cost is larger than a given threshold ϵ . As $g(n)$ is a monotonically increasing function (cost of actions are positive), the algorithm is optimal. Whenever the algorithm returns a goal, it is the one with the lower cost in the open list. Given C^* the cost of the optimal solution, its time and space complexity is $O(b^{1+\lceil C^*/\epsilon \rceil})$ [133].

- **Bidirectional search:** It is based on running two parallel searches, one forward from the initial state and one backwards from the goal state. Backward search might not be that easy if some actions are not reversible or there are several goal states, in which case the branching factor would be much larger. Instead of testing whether a node is a goal, it is checked whether the frontiers of the two searches intersect. It is complete and optimal when BFS is used in both searches. Time and space complexity is just $O(b^{d/2})$.

Table 2.3 compares different properties of the uninformed algorithms presented before.

Search	Completeness	Optimality	Time complex.	Space complex.
Depth-first (DFS)	Yes ¹	No	$O(b^m)$	$O(bm)$
Depth-limited	Yes ^{1,2}	No	$O(b^l)$	$O(bl)$
Iterative Deepening (IDDFS)	Yes ¹	Yes ⁴	$O(b^d)$	$O(b^d)$
Breadth-first (BFS)	Yes ³	Yes ⁴	$O(b^d)$	$O(b^d)$
Uninformed Cost (UCS)	Yes ^{3,5}	Yes	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^{1+\lceil C^*/\epsilon \rceil})$
Bidirectional	Yes ^{1,6}	Yes ^{4,6}	$O(b^{d/2})$	$O(b^{d/2})$

Table 2.3.: Comparison of uninformed algorithms where: (1) If the algorithm avoids repeated states and redundant paths and the search space is finite; (2) Limit depth l larger than depth of solution d ; (3) If finite branching factor; (4) All actions have same cost; (5) Complete if $stepcosts \geq \epsilon$; (6) Both directions use breadth-first search.

Informed Search

Informed search strategies can find solutions more efficiently than uninformed. **Best-first** search is the general approach used by almost all informed algorithms. The node with the best value for a given **evaluation function** $f(n)$ will be the next to be expanded. The only difference with respect to uniform-cost algorithms is the use of $f(n)$ instead of $g(n)$ to order the open list. Most evaluation functions use a heuristic function, defined as follows.

Definition 28 (Heuristic function ($h(n)$ or h -value)) *Estimated cost of the cheapest path from the state of node n to a goal state.*

Three of the most relevant informed search algorithms are:

- **Greedy best-first search:** Its evaluation function is $f(n) = h(n)$, that is, it tries to expand the node which is estimated to be closer to the goal. It is not optimal and, as DFS, is not complete because it might get trap in infinite loops.
- **A* search:** Its evaluation function is $f(n) = g(n) + h(n)$, where $g(n)$ gives the cost from the root node to another node n in the frontier and $h(n)$ the cost from n to a goal. Hence, $f(n)$ estimates the path cost from the initial state to the goal through n . A* is identical to Uniform-cost search except that it uses f instead of g . Some definitions are required to characterize the completeness and optimality of A*.

Definition 29 (Admissibility) A heuristic is admissible if it never overestimates the cost of reaching the goal, i.e. the cost it estimates is not higher than the lowest possible cost from the current point in the path [133].

Definition 30 (Consistency) For every node n and every successor n' of n generated by an action a , the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n :

$$h(n) \leq c(n, n') + h(n')$$

Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is. Every consistent heuristic is also admissible but the opposite does not hold.

A* is optimal in a tree state-space if $h(n)$ is admissible, while for a graph state-space $h(n)$ must be consistent. During search, A* add nodes in concentric bands of increasing f-cost (the **fringe**). With uniform-cost the fringe will be circular while with A* it will stretch toward the goal state, becoming narrower around the optimal path with more informative (lower error) heuristics. If C^* is the cost of the optimal solution path, then it is granted that:

- A* will expand all nodes with $f(n) < C^*$.
- A* might expand some of the nodes right on the “goal fringe” ($f(n) = C^*$) before selecting a goal node.
- A* will not expand any node with $f(n) > C^*$.

In consequence, A* is **optimally efficient** for any given consistent heuristic. That is, no other optimal algorithm expands fewer nodes than A* (except through tie-breaking among nodes with $f(n) = C^*$). In case an algorithm does not expand all nodes with $f(n) < C^*$, it might miss the optimal solution.

The time complexity depends on the heuristic and the state-space. Two concepts are required to analyse it as explained in [133]:

- **Absolute error:** $\Delta \equiv h^* - h$ where h^* is the actual cost of getting from the root to the goal.
- **Relative error:** $\epsilon \equiv (h^* - h)/h^*$.

In the best case, time complexity is polynomial when the search space is a tree, there is a single goal state, and $\Delta \leq O(\log(h^*))$. However, in most practical cases it is exponential in the length of the solution even if ϵ is constant.

Similarly to Breadth-first search (from which A* is a variant), worst than the time complexity is the space complexity, exponential to the length of the optimal path. Assuming that planning on-ground is accomplished in powerful machines, A* is the most suitable algorithm for on-ground planning in the rescue and Mars scenarios.

- **Simplified Memory-bounded A* (SMA) search:** The memory problems of A* might not be that critical if the algorithm is run in a server with plenty of resources. However, if it is used in the on-board computer of a robot, where resources are scarce, the situation changes. To solve this problem SMA* limits the amount of memory used while preserving the properties of A*.

Just like A*, it expands the best leaf node according to the heuristic until the memory is full. At this point, SMA* drops the leaf node with the worst f – value, but will preserve its f – value, so that SMA* can regenerate the subtree if it turns out in the future to be the most promising path. It is easier to understand how the algorithm works with the example in Figure 2.4.

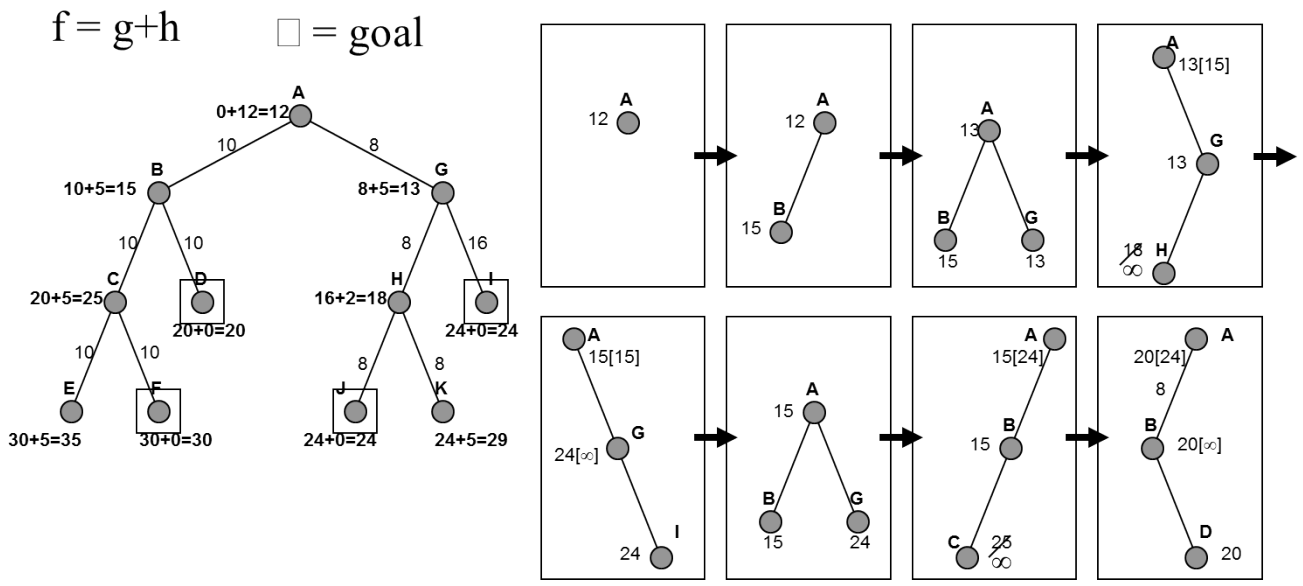


Figure 2.4.: Progress of SMA* with a memory size of three nodes. Each node is labelled with its current f - cost. Values in parentheses show the value of the best forgotten descendant (Courtesy of Stuart Russell and Peter Norvig).

SMA* is complete if the depth of the shallowest goal node is less than the memory size expressed as number of nodes. It is optimal if the optimal solution is reachable, otherwise it returns the best reachable solution. It is a very robust algorithm for finding optimal solutions, particularly when the state-space is a graph, step costs are not uniform and node generation is expensive compared to the overhead of maintaining the frontier and the explored set. These properties make SMA* particularly suitable for on-board replanning in the reference scenarios.

Table 2.4 compares different properties of the informed algorithms presented before.

Search	Completeness	Optimality	Time complex.	Space complex.
Greedy best-first	No	No	$O(b^m)$	$O(b^m)$
A*	Yes	Yes	$O(b^\Delta)$	$O(b^\Delta)$
SMA*	Yes ⁴	Yes ⁵	$O(b^\Delta)$	

Table 2.4.: Comparison of informed algorithms.

Heuristics

The efficiency of informed search algorithms depends to a great extent on the heuristics used. A heuristic typically represents a relaxation of the problem as it ignores some characteristic or constraint. It is easier then to solve the problem taking some assumptions about the consequences of the actions in the future.

The type of planner defines the type of heuristics to be used: node selection for state-space, distance between goals for GraphPlan or flaw selection and resolver selection for plan-space.

Most of the research on heuristics is done for the state-space. Four groups have been identified and described in [87]:

- **Critical Paths:** Approximate the cost of a goal set by the most costly subgoal. It is admissible because it is always more difficult to achieve larger subgoals. The most relevant example is h^m [84].
- **Delete relaxation:** Estimate the cost of reaching a goal state by considering a relaxed task Π^+ derived from the actual planning task Π by ignoring all deleted effects of operators. Some examples are h^+ (admissible) [91], h^{add} (not admissible) and h^{max} (admissible) [19].
- **Abstraction:** Abstraction heuristics map each state s of Π to an abstract state $\alpha(s)$. The heuristic value $h^\alpha(s)$ is then the distance from $\alpha(s)$ to the nearest abstract goal state. Some examples are Pattern Databases (PDBs) [58] or Merge-and-Shrink [88].
- **Landmarks (LM):** A fact-landmark for a state s is a fact that is true at some point in every s -plan, while the more recent disjunctive action-landmarks are sets of actions of which at least one is part of every s -plan. Some examples are h_{LA} [97] or h^{LM-cut} [87], both admissibles.

Heuristics in plan-space are not only required to choose the next node to expand, but also the next flaw (iterative repair approach) and the next resolver. However, they have not been so deeply studied as for state-space.

Some of the heuristics identified in [48, 49] to select the next node to expand are:

- **Fewest Threats First (FTF):** Order the nodes in the open-list by the ascending number of threats to be resolved. It aims at finding solutions quicker.
- **Most Threats First (MTF):** Aims at driving the search down through the hierarchy to find the subplans causing conflicts with others so that they can be resolved more quickly.

Regarding the selection of the next flaw, a formal definition will be first provided, followed by a list of related heuristics.

Definition 31 (Flaw) *A flaw in a plan can be: (1) Open-Goal: Goal which constraints have not been yet satisfied; (2) Threat: Action in the plan that presents conflicts.*

Some heuristics used to select the next flaw to be resolved are:

- **Zero-commitment:** Gives top priority to unmatchable open conditions (enabling the elimination of the plan). As a variant, ZLIFO gives next the priority to goals that can only be achieved uniquely, and otherwise uses LIFO prioritization.
- **Least-commitment:** Selects the flaw which generates the fewest refined plans.
- **Fewest alternatives first (FAF):** This is the most commonly used. It selects the flaw having the smallest number of resolvers in order to limit the cost of an eventual backtrack.

FAF has been proved useful for different kinds of applied planners such as HTN [149, 148, 120] and timeline-based [49].

The development of advanced heuristics is outside the scope of the thesis. However, some heuristics must be implemented to get real measurements of the planner performance.

2.4.2 Forward-Chaining Planners

One of the most successful approaches to modern planning is forward search. It starts with the (fully) defined initial state and explores the search space by iteratively choosing a state from the open-list, expanding it in successors added to this open-list until either the goal state is reached or the open-list is emptied. Forward-chaining (FC) planners differ mainly in the search algorithm used to decide the expansion strategy, and heuristics (if required by the search algorithm) used to evaluate the cost of the nodes. One of the main advantages of FC is that it can use very informative heuristics, as it always expands nodes which states are fully defined.

Some of the most influential FC planners in the deterministic satisficing track (probabilistic or optimal tracks are not relevant for this thesis) are the Fast Forward (FF) planner[91], winner of the AIPS-2000 planning competition, which started a period of dominance by heuristic search systems that last to the present, Fast Downward[86] and LAMA[128].

Two planners are examined in depth. The first, Fast Downward, has played an extremely relevant role in this thesis, not only for the search strategy followed during unfolding, but for other advanced concepts such as multi-queue heuristic search, deferred heuristic evaluation, etc. The second, TLPlan, combines forward-chaining with temporal reasoning, being therefore highly relevant for QuijoteExpress.

Fast Downward [86]

Fast Downward is a FC heuristic search planner, winner of the satisficing track at IPC 2004. Fast Downward solves a problem in three phases:

1. The *translation* component is responsible for the transformation of the PDDL problem into a *SAS⁺* problem.
2. The knowledge compilation generates four kinds of data structures. *Domain transition graphs* encode how state variables can change their values. The *causal graph*[90] represents the hierarchical relations between tasks, used later on to compute heuristic functions. The *successor generator* is a data structure for determining the set of applicable operators in a given state. Finally, the *axiom evaluator* is a data structure for computing the values of derived variables.
3. Search: In charge of doing the planning. It is based on two different varieties of the greedy best-first search algorithm: single and multi-heuristic (causal graph and FF).

Fast Downward incorporates two search enhancements. *Preferred operators* are used to give special priority to those transitions computed by the causal graph heuristic or helpful actions computed by the FF heuristic. Deferred heuristic evaluation assigns to each node the heuristic value of its predecessor and postpone its own evaluation to the time when the node is expanded. It is further analysed in Section 4.3.5.

OPTIC [10]

The OPTIC planner family represents the most powerful representation of classical temporal planning. It inherits the knowledge obtained during the development of previous temporal solvers such as LPRPG, CRICKEY, Colin or POPF.

OPTIC is capable of handling PDDL 3/+. It is based in the so called *partial order planning forwards*. Rather than enforcing a strict total order, OPTIC builds a partial-order based on the facts and variables

referred to by each step. OPTIC allows the representation of trajectory preferences (something similar to the intermediate goals that can be represented in timeline-based planners), soft constraints and the use of continuous cost functions that allow the definition of sophisticated temporal metrics beyond the makespan.

2.4.3 HTN Planners

There are two types of planners according to the control knowledge used: (1) Those using some sort of temporal formulas to tell which part of the search space should be avoided, including TLPlan, TalPlan and AP²; (2) Those using HTN methods such as SHOP2 and SIPE-2. HTN planners search space consists only of those nodes that are reachable using their methods, whereas TLPlan can explore any part of the search space that avoids the “bad” states and their successors. It is hard to say which type of control knowledge is more effective. As stated in [4], the two types are useful in different situations and combining them might be an interesting topic for future research.

SHOP2 [116]

SHOP2 (Simple Hierarchical Ordered Planner) is an HTN, ordered task decomposition planning system. Like most other HTN planners, SHOP2 is “hand-tailored”: its planning engine is domain-independent, but the HTN methods may be domain-specific, and the planner can be customized to work in different problem domains by giving it different sets of HTN methods.

Like its predecessor SHOP, SHOP2 is a sound and complete. It generates the plan steps in the order in which they will be executed. Thus, SHOP2 knows the current state at each step of the planning process.

SHOP2 have several capabilities that go beyond those of a traditional HTN planner:

- Allows each method to decompose into a partially ordered set of subtasks, and allows the creation of plans that interleave subtasks from different tasks.
- Incorporates many features from PDDL, such as quantifiers and conditional effects.
- Can handle temporal planning domains translating temporal PDDL operators into SHOP2 operators.

SIPE-2 [158, 160]

SIPE-2 (System for Interactive Planning and Execution) is a general-purpose, performance-oriented, HTN-based planning and execution system. It has been employed in numerous problems such as air campaign planning, oil spill response, production line scheduling, etc.

It plans hierarchically using different levels of abstraction. Given an arbitrary initial situation and a set of goals, SIPE-2 either automatically or under interactive control combines operators to generate plans to achieve the prescribed goals. Efficiency has been one of the primary goals in the design of SIPE-2, which includes many heuristics for reducing computational complexity. Unlike expert systems, the SIPE-2 architecture is capable of generating a novel sequence of actions that responds precisely to the situation at hand. In addition, SIPE-2 implements an execution/monitoring system that accepts new information about the world and modify the plan minimally according to this information.

2.4.4 Timeline Frameworks and Planners

Frameworks are toolkits that facilitate the development of P&S solutions and typically provide fully implemented planners. Table 2.5 compares some of the characteristics of the most relevant frameworks for QuijoteExpress.

Planner	Comm	TL	HTN	PG	Replan	MI	OBOG	Heur
ASPEN	Early	✓	✓	✗	✓	✓	✓	✓
EUROPA	Least	✓	✓	✓	✗	✓	✓	✓
IxTET	Least	✓	✗	✓	✗	✗	✓	✓
APSI	Least	✓	✓	✓	✗	✗	✓	✓

Table 2.5.: Planners comparison. Going from left to right, the columns are: (Comm) Commitment strategy; (TL) Timeline-based; (HTN) Hierarchical Task Networks; (PG) Partially grounded; (Replan) Replanning capabilities; (MI) Mixed-initiative; (OBOG) On-board and On-ground support; (Heur) Heuristic-based search.

HSTS [111, 112, 108], CAIP / EUROPA2 [21, 22, 71]

HSTS (Heuristic Scheduling Testbed System) is one of the first timeline-based planners in which many others such as Europa, ASPEN and APSI have been inspired. It consists of a heuristic chronological-backtracking search operating over a constraint-based temporal database using iterative repair. It represents high-level state variables as timelines. The planner tries to place the tokens such that all the resource and temporal constraints are met. The way HSTS defines time windows in order to react to unexpected events has inspired the concept of Consistency Horizon of QuijoteExpress. However, HSTS does not distinguish among activities, states, and resources, which is a really important aspect in QuijoteExpress.

The CAIP (Constraint-based Attribute and Interval Planning) [71] framework, based on HSTS is oriented to the design of complex concurrent systems such as spacecraft. It is based on two techniques: STN (Simple Temporal Network) and CSPs (Constraint Satisfaction Problems) and presents several similarities with ASPEN and APSI, two timeline-based systems discussed later on this section.

In CAIP, a system is modelled as a list of components called **attributes**. An attribute is defined as a set of states, e.g. a Camera with the states OFF, ON and TakingPicture. An interval contains a start time, end time and the attribute instance to which it applies. In consequence, each attribute can only take one value at any given time.

Domain **constraints** are used to describe the necessary conditions under which an interval (state with temporal extent) can hold in a valid plan. The set of constraints associated to each interval is called **configuration**.

A planning domain D is a tuple (I, A, R) , where I is a set of intervals, A is a set of attributes, and R is a set of configuration rules.

A planning problem instance is an **incomplete plan**, and the problem is to turn it into a valid and complete plan. Unsequenced intervals are part of an incomplete plan; assertions that some activity takes without justifying them.

A **candidate plan** P_C is a mapping from attributes to sequences of intervals and a set of non-sequenced intervals. A **plan extension** of P_C is a plan P that preserves P_C adding some new intervals. A **valid plan** is a candidate plan that satisfies all planning domain constraints.

The initial plan is modified through a number of steps until it becomes a valid, complete plan. A plan can be modified by means of:

- **Restriction:** Reduces the number of options remaining for existing decisions, like adding a new decision (unsequenced interval) or reducing the number of remaining options (ground variables or reducing the domain of the variable).
- **Relaxation:** Do the contrary.

The traditional definition of a complete valid plan requires that all of the constraints are satisfied, all intervals sequenced and that the plan has been fully specified to the end of time. However, this is an unnecessarily restrictive definition. In many applications, the Executor can fill in unspecified parts of the plan. Those flexible plans that satisfy a given set of completeness requirements are referred to as **sufficient plans**, a very relevant concept that will be extended in QuijoteExpress.

There are three differences between candidate plans from solutions: (1) There may be unbound variables; (2) There may be unsequenced intervals which have to be scheduled onto the appropriate attribute; and (3) Some applicable compatibilities may not be yet satisfied.

EUROPA (1 and 2) are planners based on the CAIP framework. They have been applied to several domains, including multiple satellite scheduling, planetary rover operations, automated flight planning, Deep Space 1 mission, etc.

EUROPA2 has a powerful declarative modelling language called NDDL (New Domain Definition Language). A planning domain D in NDDL is represented by the following elements:

- A set of timelines: $T = \{T_1, T_2, \dots, T_n\}$, which are essentially variables capturing the evolution of the attributes over time.
- A set of mutually exclusive activities associated with each timeline.
- A conjunction of temporal constraints associated with each activity.

The above representation differs from a PDDL domain description in two aspects: It uses a variable/-value representation (timelines/activities) rather than a propositional representation; There is no concept of state or action, only of intervals (activities) and constraints between those activities.

A planning problem P is represented by a pair $P = \{H, I\}$ where:

- $H \in \mathbb{N}$ is the planning horizon that marks the time boundary of the activities in the plan to the interval $[0, H]$.
- I is the initial configuration represented by a set of activities placed on their corresponding timelines. For each activity, it is possible to fix the start and end time or leave it floating on the timeline. The initial configuration corresponds to both the initial state and the goal state.

The planning algorithm at the core of EUROPA2 can be thought of as an instance of the plan refinement search; given a domain D and a problem P , the algorithm starts from the initial configuration I and incrementally refines it by adding activities to the timelines, ordering those activities and binding variables until a final consistent configuration is found. There are three types of flaws, each solved in different ways:

-
- Open condition flaw: An activity in the plan requires the support of other activities, still not included in the plan.
 - Ordering flaw: An activity is placed on a timeline and an ordering is required for the activity with respect to the other activities already on that timeline.
 - Unbound variable flaw: A variable that has not yet been instantiated appear in the plan.

The basic algorithm in EUROPA2 is a depth-first search characterized by flaw selection (no backtracking), flaw resolution (backtracking) and constraint propagation steps. Operations of plan refinement are interleaved with constraint propagation on the temporal network underlying the current partial plan.

ASPEN-CASPER [75, 43]

The Automated Scheduling and Planning ENvironment (ASPEN) is one of the most successful planning systems used in space. It has been used in several missions/applications including EO-1, Citizen Explorer 1 (CX-1) [161], CLARATy, CX-1 [66] Planetary Rover Operations, Multi-rover Integrated Science Understanding System (MISUS) and more. Moreover, it is the closest approach from a theoretical point of view to QuijoteExpress / APSI among all the tools showed along this section.

ASPEN is an object-oriented, reconfigurable framework, capable of supporting a variety of P&S applications including spacecraft operations, mission design, surface rovers and ground antenna utilization.

ASPEN consists of:

- A language called ASPEN modelling language (AML) used to describe the domain model.
- A constraint management system for representing and maintaining resource constraints and activities.
- A set of search algorithms for planning.
- A language for representing plan preferences and optimizing these preferences.
- A temporal reasoning system for representing and maintaining temporal constraints.
- A graphical interface for visualizing.

The job of the planner/scheduler is to accept high-level goals and generate a set of low-level activities that satisfy the goals, do not violate any of the spacecraft flight rules or constraints, and optimize the quality of the plan. ASPEN uses an early-commitment, local, heuristic, iterative search approach.

Spacecraft models are developed with AML by means of seven different model components:

- Activity: Occurrence over a time interval that in some way affects the spacecraft. It can represent anything from a high-level goal or request to a low-level event or command. Once the types of activities are defined, specific instances can be created from the types. For example taking a picture with a camera represents an activity. An activity has a set of parameters, parameter dependencies, temporal constraints, reservations and decompositions. All activities have at least three parameters: start time, end time, and duration. There is also at least one parameter dependency relating these three parameters. In addition, all activities have at least one temporal constraint that prevents the activity from occurring outside of the planning horizon.

-
- **Parameter:** Variable with a restricted domain. For example, the starting time is a parameter for the action take picture.
 - **Parameter dependency:** Functional relationship between two parameters. For example, the completion time for an activity is a function (the sum) of the start time and the duration. A more complicated dependency might compute the duration of a spacecraft slew from the initial to the final orientation.
 - **Temporal constraint:** Relationship between the start or end time of one activity with the start or end time of another activity. One might specify, for example, that an instrument warming activity must end before the start of an activity that uses the instrument.
 - **Reservation:** Requirement of activities on resources or state variables. For example, an activity can have a reservation for 10 watts of power.
 - **Resource:** Represents the profile of a physical resource or system variable over time such as power or memory. In ASPEN, a resource can either be depletable or non-depletable (see 2.4.4). A resource can be assigned a capacity, restricting its value at any given time.
 - **State variable:** Represents the value of a discrete system variable over time. The set of possible states and the set of allowable transitions between states are both defined with the state variable. An example of a state variable is a camera that has three states: OFF, ON and TakingPicture.

Activity hierarchies can be specified in the model using decompositions in a similar way to HTN planners (see 2.3.2). A decomposition specifies how to accomplish a high-level activity by means of a set of subactivities along with temporal constraints between them. The subactivities may have in turn their own decompositions. In addition, an activity may have multiple decompositions to choose from.

A conflict is a violation of a parameter dependency, temporal or resource constraint. For each conflict type, there is a set of repair methods. The search space consists of all possible repair methods applied to all possible conflicts in all possible orders. The approach of ASPEN to searching this space is based on a technique called **iterative repair** [166, 124]: First the conflicts in the schedule are detected, the repair algorithm selects a conflict (one at a time) and then selects a repair method. The process is repeated until no conflicts exist, or a user-defined time limit has been exceeded.

To achieve a higher level of responsiveness in a dynamic environment, CASPER (for Continuous Activity Scheduling Planning Execution and Replanning) is used [40]. Besides the conventional inputs, CASPER has the current plan. The planner is responsible for maintaining a consistent, satisficing plan with the most current information according to the following cycle: Changes in initial/goal state; Propagate effects of these changes; Invoke repair algorithms to remove conflicts.

ASPEN also facilitates reasoning about plan quality [123]. It adopts a local, early-commitment, iterative approach to optimization. During iterative optimization, low scoring preferences are detected and addressed individually until the maximum score is reached or a user-defined time limit has been exceeded in a similar way as plan repair is achieved.

APSI [35, 72]

APSI (Advanced Planning and Scheduling Initiative) is the software instantiation of **TRF** (Timeline Representation Framework), a theoretical framework aiming to provide the foundations to build Artificial

Intelligence Planning and Scheduling technologies with timelines. TRF plays a critical role in this thesis, therefore the concepts presented in this section will be used throughout the whole dissertation.

Two inputs are required to perform planning: the domain and the problem.

Definition 32 (Domain) A domain D is used to formally describe the world about which it is required to generate plans. A TRF domain is constituted by a set of **components** and a domain theory that states the relations among the values of the components.

Definition 33 (Component) The world is modelled as a set of entities, called **components** in TRF, whose properties can vary in time according to some internal logic or as a consequence of external inputs. A component is a reasoner that manages timelines and assertions that specify the properties of the timeline. TRF contains three types of components: state variable, consumable resource and reusable resource.

Before describing the different types of components, the concepts of decision and relations must be introduced.

Definition 34 (Behaviours and Decisions) A **behaviour** is defined as a tuple $\sigma = \langle T_\sigma, N_\sigma \rangle$ where T_σ is a finite set of ordered time instants in H and N_σ is an assignment of values to the time instants in T_σ . A **decision** is defined as $d = \langle \tau_d, v_d \rangle$ where $\tau_d = [t_s, t_e)$ is the time frame, such that $t_s, t_e \in H, t_s < t_e$ and $v_d \in V_C$ is a value of V_C , the set of values applicable to a component C . Only some behaviours are valid for a state variable, which are defined by means of **transition functions**.

Definition 35 (Relation) A relation is used to define some constraint affecting one or more values. Relations in TRF can be organized in two groups:

- A n -ary temporal relation can be generalized as $f_{temp} = \langle type, [lb, up]_0^3, t_f = \langle t_1, \dots, t_n \rangle \rangle$ where $type$ is a symbol that identifies an Allen's temporal relation [3], $[lb, up]_0^3$ are time frames which quantity (zero to three) depends on type and t_f containing the list of time instants t_i affected by the relation. Along the thesis, a more relaxed notation of temporal relation will be used in which the list of time instants is replaced by the list of decisions to which the time instants belong, formally ($f_{temp}^{type}(d_1, d_2)$) where $scope(f) = d_1, d_2$ contains the list of two decisions involved in the relation. Even though these relations are n -ary according to the formal definition, STP problems allows only binary relations. As an example, *FindRoute* before $[0, 5]$ *TraverseCycle* specifies that *FindRoute* will happen between $[0, 5]$ time units before *TraverseCycle*.
- A n -ary data (or parameter) relation can be formally defined⁶ as $f_{param} = \langle type, t_f = \langle t_1, \dots, t_n \rangle \rangle$ where $type$ defines the type of the relation and t_f is the scope containing the list of terms or predicates affected by the relation. As for temporal relations, a the following (more relaxed) notation will be used: $f_{param}^{type_p}(t_1, t_2, \dots, t_n)$ where $type_p$ is a linear parameter constraint such as *equal*, *bigger_than* or a linear function and $scope(f) = t_1, t_2, \dots, t_n$ the parameters involved. An example of data relation is: $pointing_{camera} = position_{target}$.

The APSI framework provides several implementations of components:

Definition 36 (State Variable) Component used to represent systems that can be modelled as state machines such as the locomotion system showed in Figure 2.5.

A state variable C_{SV} have a number of behaviours belonging to the co-domain $D_{C_{SV}} = \langle S, \mathbb{D}, V \rangle$ where S is a finite set of symbols, \mathbb{D} a finite set of parameter domains and V a finite set of values which

⁶ This definition is a generalization as the definition changes with the type.

represent the different states. A value $v \in V$ has associated a **predicate** and a minimal and maximal allowed temporal duration. A predicate is defined as $\text{pred}_v = \langle s_v \in S, T_v = \langle t_{v_1}, \dots, t_{v_m} \rangle \rangle$, where the symbol s_v represents the name of the value and T_v is the set of parameters (terms) of v , each having a specific domain $D(t) \in \mathbb{D}$. The domain of a parameter can be either an enumeration or a bounded integer. When a value v is **instantiated** into a decision d_v , it is assigned a unique identifier that allows to distinguish it from other decisions with the same predicate, and a time frame $\tau_d = [t_s, t_e)$. The parameters must be also instantiated, getting as well a unique identifier. The application of a decision to a state variable represents the choice of a value over a specific time interval.

Definition 37 (Resource Component) A resource is any physical or virtual entity of limited availability such as battery or memory. The concept of limited availability is defined by means of minimum and maximum bounds: a temporal function representing a behaviour of a resource is consistent if it is always between the allowed bounds in the interval H . There are two classes of resources: **reusable** C_R and **consumable** C_C which difference stems in the type of decision applicable. In both cases, a decision captures the concept of resource usage, but for a reusable resource a decision represents an amount of resource booked on a temporal interval while for a reusable resource it is an amount produced or consumed in a time instant.

Taking as an example the FASTER Mars rover scenario⁷, its domain includes a model of the rover subsystems (e.g. locomotion, battery, communications, etc.) and external elements (e.g. communication windows). Figure 2.5 shows locomotion system of the primary rover modelled as a state variable with four nominal states: idle, traversing, turning or initializing.

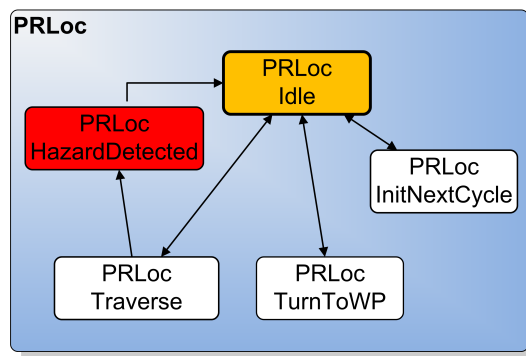


Figure 2.5.: State Variable component representing the locomotion subsystem of the primary rover in the FASTER scenario.

A behaviour for the locomotion component could be:

$$\sigma = \langle T_\sigma = t_1, t_2, t_3, N_\sigma = \text{Idle}(), \text{Traverse}(x_o, y_o, x_d, y_d), \text{Idle}() \rangle$$

where t_1, t_2, t_3 are time instants, *Idle* and *Traverse* are value symbols and x_o, y_o, x_d, y_d are integer parameters indicating the original and intended destination position for the rover.

Definition 38 (Timeline) A timeline is a finite set of variables representing ordered points in a temporal interval H and variables representing values related to them, that is, a set of partitions of H into intervals $\tau_i = [t_i, t_{i+1})$ and associated values $v(\tau_i)$. State variable timelines assign to time points states of the state variable, possibly with constraints among their parameters, while resource timelines assign to time points integer parameters, possibly with linear constraints among them. A **component timeline** take

⁷ This scenario is fully described in Chapter 6.

values in a set of possible behaviours (set of valid assignments of values to time instants in the interval $H = [t_0, t_H)$) while a **domain timeline** is a variable taking values in a set of **domain evolutions** (tuple of component behaviours).

In real domains, the different subsystems influence each other's behaviour. Hence, it is needed to define valid patterns to represent the evolution of the system and the effect of decisions. This is accomplished by means of synchronizations.

Definition 39 (Synchronization) *A synchronization is a rule between a reference value (that triggers the application of the synchronization) and a target network composed of values and some n-ary relations on time frames over the values describing the effects of triggering the reference.*

For example, *Communicate DURING Window*, is a synchronization which says that a communication with the relay satellite can only happen when the satellite is visible in the horizon, that is, whenever there is a communication window.

Once the domain is defined, the user can create problems for this domain.

Definition 40 (Problem) *The problem is modelled by stating a set of behaviours describing the initial status for each component, a set of behaviours and decisions describing the goals and a domain theory DT. The problem is represented by means of a Decision Network.*

Provided a domain and a problem, it is possible to use APSI planners in order to extract valid plans. A general solving process with timelines is an iterative process based on the following steps: (1) Analyse the current status of the timelines and the decisions against the goal conditions and the domain theory and then produce a list of flaws; (2) Analyse and heuristically prioritize the flaws; and (3) On the basis of the flaw and the status of the timelines, select a planning or scheduling step to be applied in order to remove the flaw and produce a solution. A step can consist on the addition/deletion of a decision or relation to the decision network.

Definition 41 (Solution) *A solution of a problem is an assignment of values to the elements (values and parameters) such that all constraints posted are satisfied.*

AP² [33] is a planner based on the APSI-TRF technology used for planning and re-planning activities in GOAC [30]. The planner can be invoked by a deliberative reactor that provides problems in the form of user goals (planning) or plans to be fixed (replanning) and produces a number of solutions in the form of flexible timelines.

2.4.5 Expert Systems for Rovers

There is a wide variety of autonomous software dedicated to solve specific problems. Some examples used in space missions are opportunistic science [29], improved communications [34, 31], autonomous inspection and servicing (AIS) [118], autonomous rendezvous and docking (AR&D) [26], planetary entry, descent and landing (EDL) [121], human-robot interaction (HRI) [55] and a long etc.

In the specific case of planetary rovers, the extension of MER's nominal mission, initially planned for 90 sols and running for more than 10 years in the case of Opportunity, allowed researchers to further develop and infuse autonomous technologies. Autonomy has been used in both on-ground and on-board segment, but it could be arguably said that the main effort has been focused on-board, as depicted in [135, 85, 67]. Even though the next subsections summarize the autonomous software used for MER/MSL missions, there are many other tools developed for research or testing purposes.

Autonomous Navigation - Autonav

The mobility flight software of MER / MSL provides a list of different navigation modes for the rover [17, 16]. One of them, the autonomous navigation with hazard avoidance (Autonav), is with no doubt the most used autonomous software on-board. Autonav combines global path planning (Field D*) with local hazard avoidance (GESTALT) to autonomously navigate around complex terrain [25]. During each cycle, the rover acquires stereo images, assess the terrain, selects a safe drive arc and finally drives along the arc.

Visual Target Tracking - VTT

Visual target tracking (VTT) is fully integrated into the existing MER flight software. VTT can run in any combinations of rover driving: blind driving (with IMU-based estimator), VO (visual odometry) and/or Autonav, enabling the rover to approach a designated target 10 to 20m away within a few centimetres error [98].

Autonomous Instrument Placement - Autoplace

MER has a set of instruments installed in the turret of its robotic arm. Normally, the target for an instrument placement is identified in stereo imagery acquired at the rover's current location, that is, no driving occurs between the imaging and the instrument placement. This allows rover operators to manually create a sequence of commands which is verified both in simulation and by the on-board rover software. Some safety checks however require knowledge of the terrain: checking for collisions between the arm and the surface or ensuring that the instrument placement does not generate high loads. A sol of operations can be eliminated by moving the target selection, trajectory generation, and terrain collision analysis into the on-board software. This software is called Autonomous Instrument Placement (AutoPlace) [85].

Dust Devil / Cloud SPOTTERS

Dynamic atmospheric phenomena in Mars include dust devils and clouds. Traditionally, campaigns to observe them have been conducted by collecting a set of images at a fixed time, pre-specified in the command sequence, and then downloading the image set. This process is highly inefficient as most of images contained no interesting events. A new approach consists on analysing images on-board to identify the presence of clouds and dust devils. By selecting only those images that capture the events, more images can be stored on-board resulting in a much greater science return. Both algorithms were integrated with the MER flight software and have successfully identified several events with a 93% accuracy for the cloud detection and 85% for the dust devils [28].

Automated Science Targeting - AEGIS

The Autonomous Exploration for Gathering Increased Science (AEGIS) system enables automated data collection by planetary rovers. AEGIS software was uploaded to the Opportunity rover in 2009 and has successfully demonstrated automated on-board targeting based on scientist-specified objectives. Prior

to AEGIS, images were transmitted from the rover to the operations team on Earth; scientists manually analysed the images, selected geological targets for the rover's remote-sensing instruments, and then generated a command sequence to execute the new measurements. AEGIS represents a significant paradigm shift: it uses scientist inputs to automatically select and sequence on-board high-quality science targets with no human in the loop, which is particularly useful for narrow field-of-view instruments such as the MER Mini-TES spectrometer, the MER Panoramic camera, and the 2011 Mars Science Laboratory (MSL) ChemCam spectrometer [67]. This approach enables opportunistic science capabilities on-board that can be considered equivalent to an E4 level of autonomy.

On-ground Plan Generation - MAPGEN, PSI and MSLICE

There is a number of tools used to generate the commands that will be uplinked to the spacecraft specially developed for Mars missions. The first was the MAPGEN system, used to build plans for MER (Bresina et al, 2005). MAPGEN assisted users, primarily the mission tactical planners and scientists, to build efficient and safe rover plans each day in a collaborative manner. The tool, however, did not have a fully integrated power planning element, but rather relied on a simplified model to do planning that had to be later on verified. PSI (Phoenix Science Interface) developed for the Phoenix Mars Lander and MSLICE (Mars Science InterfaCE) for MSL are follow-on applications (Aghevli et al., 2006).

During the 19.5 hours left to generate the tactical plan for rover missions, data from the previous sol is combined with the long-term plan to determine the scientific objectives for the next sol. Scientists are encouraged to oversubscribe resources to ensure that they will be fully utilized in the final plan. In the next step, the science observation requests are merged with the engineering requirements and a detailed plan and schedule of activities is constructed for the upcoming sol using either MAPGEN or MSLICE.

A schedule is a collection of activities planned to occur at predetermined times. Each activity has associated constraints and utilizes resources, which are modelled within the system. The planner has to integrate many different types of activities and constraints without violating any of the flight rules, like exceeding power or data limits. To support this process, MAPGEN, PSI and MSLICE use EUROPA [71] (see 2.4.4).

Once approved, the activity plan is used as the basis to create sequences of low-level commands, which coordinate on-board execution. This sequence structure is then validated, packaged, and communicated to the rover, thus completing the commanding cycle [22].

Figure 2.6 shows the role of MAPGEN in the plan generation process on-ground [107]

2.5 Plan Execution

One of the most important differences between planning for classical or robotic domains is that in the later one, the plan is meant to be executed by a physical system.

Traditionally, execution has not been so deeply studied as planning, among other reasons due to the scarce amount of platforms to test them, the absence of standards and mainly because it is considered to be much easier than planning.

Nonetheless, the proliferation of sophisticated robots in research labs showed the need to invest more efforts not only in the development of hardware, but also in the software to control them. It is precisely in this area where the appearance of ROS (Robot Operating System) has played a major role.

Definition 42 (ROS) *ROS is an open-source, meta-operating system for robots, providing services typical from an operating system such as hardware abstraction, low-level device control, implementation*

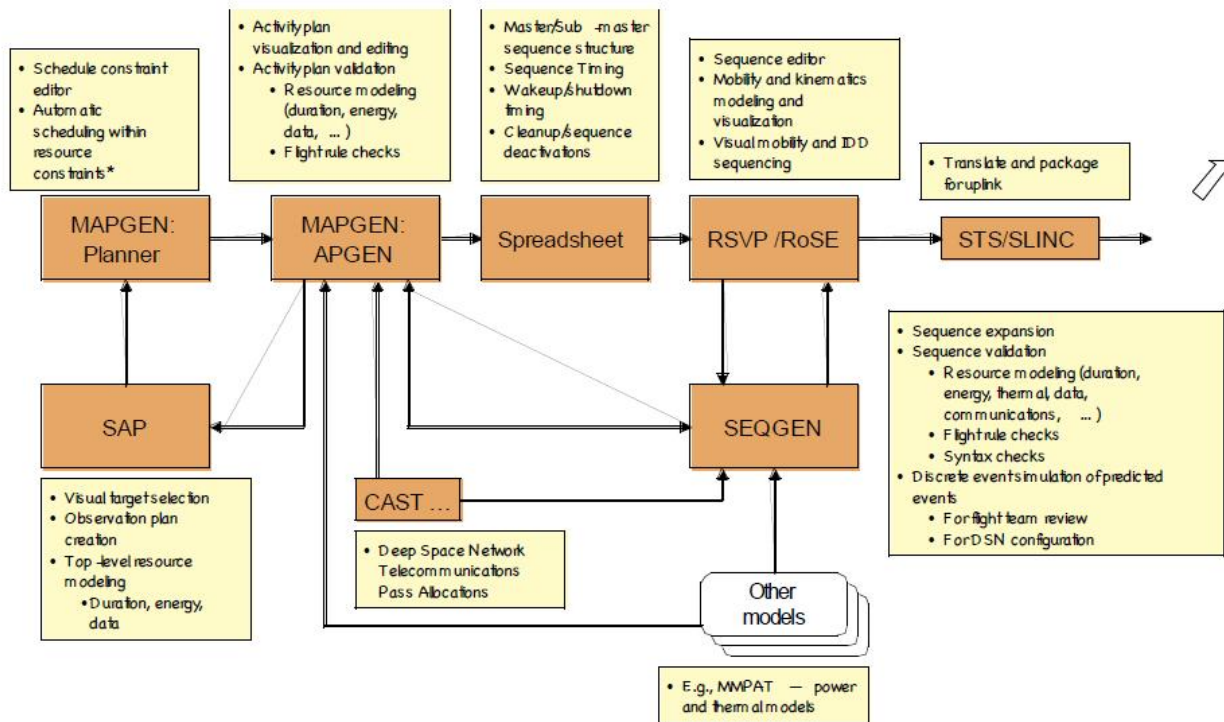


Figure 2.6.: MER Uplink process (Courtesy of J. Richard Morris and NASA-JPL).

of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

In the case of space exploration the problem is different. In spite of the large amount of satellites launched every year, none of the multiple executives developed so far [152] has managed to become a standard. The expected proliferation of (cheaper) robotic mission sharing common platforms such as the 2020 Mars mission, which will widely reuse the MSL design, can help to reuse low level software. Moreover, the irruption of ROS in the space sector [1] and its recent movement toward real-time systems [79] might also facilitate the development of an standard executive system.

This thesis presents not only a mission planner, but an entire robotic architecture, a system involving several components intended to control the robot behaviour. It is usually divided in several layers operating at different levels of abstraction and reaction-time. There are three main types of architectures:

- Deliberative: Top/down approach focused on high level control, which implies low capabilities to react in real-time to unexpected events.
- Non-deliberative: Bottom-up approach, focused in reactive behaviour (could be real-time). High level, strategic plans are difficult to implement in these systems.
- Hybrid: Present a slow, deliberative reasoning system in charge of deciding the strategic plan and a fast, reactive system in charge of reacting to unexpected changes of the environment.

In the past, three tier architectures [78, 18, 2], which divide the software in planning (deliberative), executive (reactive) and functional (control) layers, clearly dominated. However, at present two tiers [109, 155, 30] combining planning and executive in one single layer have demonstrated to be a good alternative.

Designing and implementing a wholly operational architecture is an enormous task demanding a lot of engineering, which is beyond the scope of a thesis. The aim of the architecture presented here is quite different, as it has been build to demonstrate the main building blocks of this thesis, namely the planner (Chapter 4) and executive (Chapter 5). In consequence, the following discussion focus on the different executives and their properties.

Executives are based on the principles of control theory, that is, to control the system so that its output follows a desired control signal. In opposition to deliberative systems, executives have negligible reasoning capabilities as they must behave in a very reactive manner. Execution is however a complex task due to the special properties of real-world scenarios as listed in 2.2.1. In case any of the assumptions $A0$ to $A3$ hold, then it is not possible to guarantee execution completeness as defined in Section 2.6.

An executive typically consists on two components:

- **Dispatcher:** Selects the next activity to be executed and timely sends it to the corresponding sub-system.
- **Monitor:** Tracks the execution of the activities, taking appropriate measures in case of deviations with respect to the nominal plan and reporting them back to the planner.

Executives can be classified according to different properties:

- **Feedback:** The system might provide feedback as data from sensors monitoring the system. The difference between the feedback and the expected output is the error, used to modify the input and therefore minimize the error. This type of system is called closed-loop in opposition to open-loop, which does not have (or use) any measurement of the output.
- **Linearity:** The output of some systems are governed by linear equations. Specially relevant in this category are those which output is time invariant. Otherwise, the system is nonlinear and applies to more realistic domains.
- **SISO vs MIMO:** Depending on the number of inputs and outputs, control systems range from SISO (Single-input single-output) to MIMO (Multiple-input multiple-output).

The reference scenarios presented in Chapter 1 are nonlinear and demand closed-loop MIMO systems. Closed-loop is key because it would be simply too risky for the mission to disregard the result of execution. With respect to the complexity, a robot composed of several subsystems, sensors and actuators is clearly a MIMO system.

Next sections describe some of the most relevant executives found in the literature which can be divided in two groups: those oriented to space missions and those general purpose based on ROS.

2.5.1 T-REX

Strictly speaking, T-REX (Teleo-Reactive Executive) [125, 122] is not an executive, but a hybrid systems combining goal-driven and event-driven behaviours in an unified framework based on temporal planning and the notion of **sense-plan-act** where sensing, planning and execution are interleaved.

In order to facilitate the scalability of the system, T-REX allows the partitioning of the system in subsystems according to their functionality or temporal constraints. Each subsystem is controlled by a *reactor* that encapsulates the logic of the subsystem and might contain a dedicated planner. Some reactors have more deliberative duties, therefore having larger look-ahead windows while others need to be more responsive. Reactors can synchronize among them by means of a messaging protocol for

exchanging facts and goals. All reactors share the runtime state of their subsystems for both deliberation and execution, offering a seamless integration between planning and control.

T-REX was originally developed for Adaptive Mission Control of Autonomous Underwater Vehicles. However, it can be use in any domain that demands interleaved deliberation and execution for robotic applications.

2.5.2 IDEA - Plan Runner

IDEA (Intelligent Distributed Execution Architecture) is a real-time 2-tier architecture in which the executive and deliberative layers share a single plan representation. In IDEA, a system is implemented as a set of agents, each having a model of the system (called domain), a plan database, plan runner and a variety of planners.

The core execution component of the agent is the Plan Runner. It is expected to wake up, process all received messages, call the Reactive Planner, receive termination notification from the Reactive Planner, send appropriate messages to external agents and suspend itself within the execution latency. The agent is guaranteed to operate within a well-defined real time range, a crucial condition for embedded control system.

The Plan Runner is asynchronously activated when a message is received from another agent or because of a signal triggered by an internal timer. When it wakes up, it calls a Reactive Planner in order to add the message to the plan database. In case the Reactive Planner finds a solution, the Plan Runner starts executing the procedure.

Among the key features of IDEA and by extension of its executive it is important to remark the centralized model representation and the definition of hard reaction constraints for all agents.

2.5.3 Universal Executive (UE)

Execution engine[151] for spacecraft operations. It can execute plans with variable levels of autonomy and co-ordination among various systems and human operators. Plans are expressed with Plexil[15], an executive language used to demonstrate automation technologies targeted at future NASA space missions. Applications of Plexil have included control of hardware prototypes such as planetary rovers and drills, the Habitat Demonstration Unit, and procedure automation for the International Space Station. Plexil allows to represent branches, loops, time and event driven activities, concurrent activities, sequences, and temporal constraints.

The fundamental programming unit of UE/Plexil is the Node. A node is a data structure formed of two primary components: a set of conditions that drive the execution of the node and another set which specifies what the node accomplishes after execution. A plan is a tree divided in nodes close to the root (high level nodes) and leaf nodes that represent primitive actions such as variable assignments or the sending of commands to the external system.

2.5.4 Teer

Teer, which stands for task-execution environment for robotics, is a Python library proposing the use of tasks instead of state machines to implement executives. Teer provides the following features:

- Tasks are written as sequential code in standard Python.

-
- Tasks can wait for a certain duration, for conditions to become true or for the termination of other tasks.
 - Tasks can create new tasks and kill other tasks.
 - Tasks can pause or resume other tasks.
 - Conditions are evaluated as rarely as possible, avoiding regular polling of their expressions.

Compared to state-machines, *teer* allows to maintain sequential code for sequential actions, using multiple lightweight tasks to implement parallel flows of execution. Compared to multi-threading, the cooperative aspect of co-routines removes synchronisation hazards.

2.5.5 SMACH

It is an executive based on state machines oriented to the execution of complex plans, where all possible states and state transitions can be described explicitly. The domain is modelled with Python to facilitate the process of prototyping of hierarchical state machines. During execution, SMACH allows full introspection of the state machines, state transitions, data flow, etc.

The main modelling unit in SMACH is the *container*. There are four types: *StateMachine*, *Concurrence*, *Sequence* and *Iterator*. Each container provides different execution semantics, but they can all be treated like states in other containers. A container is defined in terms of a dictionary of states. In SMACH, a *state* corresponds to the system performing some task. This is different from formal state machines, where each state describes not what the system is doing but rather describes a given configuration of the system. SMACH provides the following type of states:

- *State*: Basic type.
- *SPAState*: A state which has three pre-defined outcomes: succeeded, preempted and aborted.
- *MonitorState*: A state that subscribes to a topic and blocks while a condition holds.
- *ConditionState*: A state that executes a condition callback.
- *SimpleActionState*: A state which acts as a proxy to preemptable tasks.

Unlike PLEXIL or T-REX, SMACH is not meant to be used as a state machine for low-level systems that require high efficiency. It also falls short as the scheduling of the tasks becomes less structured.

2.6 Benchmarking Methodology

The approach to evaluate some characteristic of an algorithm is based on two different aspects: the measurement itself by means of some predefined metrics and the comparison with the results of other algorithms.

From the three properties identified in Chapter 1, only performance can be measured quantitatively while expressiveness and robustness are analysed in a qualitative style. Section 3.3 presents the theoretical results while Section 4.4 shows the performance evaluation of QuijoteExpress compared to other APSI planners.

2.6.1 Planning Performance

The performance evaluation focuses on satisficing (in opposition to optimal) temporal problems, for which some properties such as soundness and completeness must be taken into consideration. Synthetic scenarios such as those used in the IPC are not relevant because of the limited relevance given to temporal aspects (see Section 2.2.1). Instead, variations of the Mars and Rescue scenarios will be used.

Evaluating the performance of a planning system is an empirical task, often depending on the criterion of the evaluator in terms of the weight assigned to different combined metrics. As an example, in the IPC, coverage and plan quality are combined to obtain the planner ranking. Evaluating a planner in such a way is out of the scope of this thesis. The metrics considered are:

- Number of nodes expanded: Number of nodes that the planner needed to expand before returning a solution.
- Planning time: Time required for the planner to exhaust the search space (optimal) or obtain the n-first solutions (satisficing).
- Plan quality: If any criterion about quality is provided for a given problem, the quality of the different solutions is compared.

For example, timespan and/or resource consumption represent general metrics to measure the plan quality in the two reference domains [20].

2.6.2 Expressiveness

This property can be analysed from the point of view of the grammars accepted by planning language. In addition, Section 3.3.4 presents a number of concepts derived from HTLN that cannot be expressed with the regular APSI framework.

2.6.3 Robustness

This property depends on multiple factors such as the specific domain and problem. The claims regarding robustness in Section 3.3.2 are derived from the level of flexibility allowed to the planner in order to postpone assumptions and decisions in a similar way as the start/end time intervals assigned to actions, preventing to make specific assumptions until the time when the actions are due to execution.

2.6.4 Execution Power

The categorization of the different types of solutions provided in Section 2.3.4 can be reused in the context of execution to characterize the domain. The two scenarios proposed and virtually any real-world scenario will fall inside the category of weak solutions, meaning that it is not possible to guarantee the achievement of the goals under the assumptions indicated in Section 2.2.2.

The principles to evaluate an executive are radically different from those used for planning. While planning focuses on performance, this aspect is not relevant for an executive from a theoretical point of view. An executive represents a deterministic system that should always exhibit the same behaviour for the same state in opposition to search algorithms which are non-deterministic⁸. In consequence,

⁸ Do not confuse the concept of non-deterministic choices used here with non-deterministic actions used in the assumptions defined in Section 2.2.1

performance is related to low-level implementation details not relevant from a research point of view. Evaluating an executive is however possible in terms of their execution capabilities, specially in some desirable, general properties that they might offer:

- Temporal: The executive allows the execution of time-tagged tasks at due time.
- Autonomy: The executive can operate in different levels of autonomy, from E1 to E4. In consequence, it must be able to interpret time or event-tagged plans.
- Parallelism: The executive allows to execute tasks in sequence or concurrently.
- Loops: The executive allows the execution of loops. In case the task in the loop is time-tagged, a maximum duration for the loop should be provided.
- Conditional execution: The executive should be able to define guards preventing the execution of behaviours in case they are not satisfied. The guards could be time-tagged.
- Monitoring: The executive allows the definition of monitors to observe the evolution of the system. Moreover, the executive should be able to pause/stop/resume the execution based on the information coming from the monitor and the planner.
- Tightly coupled planner and executive: Planner and executive can communicate one with each other and get access to their internal state.
- Shared plan: Planner and executive will share a common plan, allowing the planner to rapidly fix a plan taking into consideration the modifications of the executive and to the executive to resume execution without intermediate transformations as soon as the planner has finished.
- Platform-independent: The executive can be used with different robots.

2.7 Conclusions

Robotics represent a complex research field which has not yet reached matureness due to the multiple scientific advancements it requires. It has not been until recently that the different pieces required to develop autonomous robots for real-world scenarios are starting to be integrated. Focusing on the level of autonomy, robots moves in the range defined by two antagonist lines: highly automated robots operating in deterministic domains such as fabrics, and highly teleoperated robots for non-deterministic environments such as space or rescue missions. Our interest however lies precisely in the middle, that is to enable highly autonomous robots in highly complex scenarios compliant with the assumptions presented above.

After a deep analysis of the state-of-the-art in planning, it is clear that this field has not been fully addressed yet. Plenty of planners such as Fast Downward or Shop/Shop2 have been used since long in classical, deterministic domains such as logistics [89] or evacuation plans [114]. Some planners such as ASPEN or EUROPA have been used for satellites domains [110, 45, 137, 93, 32] which are quite deterministic and static (assumptions A2 and A3). EUROPA has been even used to support planning generation on-ground for MER mission [22], but no planner has been used to date on-board Mars rover or rescue robot missions.

Several factors are to be blamed for this lack of autonomy. First of all, the complexity derived from the assumptions presented in Section 2.2.2 makes automated planning and execution a hard job. In addition,

the lack of standards has prevented a faster progression, even though ROS is helping to mitigate this problem. Finally, the interest for automated robots has only recently gained traction in the (planning) research community. The reasons in the space sector are different, coming primarily from the criticality of the missions, derived from its high cost, and the impossibility to recover/replace the system in case of failure, facts specially relevant in the case of Mars rovers. In this regard, planners would need to improve the performance, expressiveness and the robustness of the plans produced to be ready for such scenarios.

Modern classical planning is highly focused on the development of heuristics for sequential, forward-chaining planning. It can be arguably stated that there have been no big novelties in the field since years. Moreover, it seems that competitions such as IPC have to move towards more realistic scenarios, including for example resource consumption, intermediate goals and mainly time as mandatory features. Therefore, in terms of expressiveness and robustness, classical planning does not seem to be the more appropriate approach. Nevertheless, applied planning can highly benefit from several novelties derived from classical planning regarding search algorithms and heuristics.

With respect to applied planning, there has also been little innovation in the field, as it seems like the work is moving from the development of the planners towards the adaptation to new domains. Contrary to classical planning, where forward-chaining clearly dominates, applied planning presents a big diversity of systems, even though the timeline approach seems to be one of the most successful for temporal problems. In spite of the remarkable contribution of POMDPs and Model Checking in non-deterministic problems, they are not suitable for the Mars and Rescue scenarios. As for POMDPs, timeline planning might have utility functions (based on costs and rewards) that lead the planner towards the best solution. However, classical policies defined as pairs (state,action) are not applicable because the number of states might be infinite. At the same time the probabilities of the transitions are unknown. The equivalent of Model Checking “policies” in QuijoteExpress are the HTN methods which do not contain all the states, but just those required to achieve complex tasks. Unlike Model Checking, the application of an action in the reference domains produces one single state, the one obtained from the ideal execution of the action. It is also not realistic in real systems to apply actions without knowing, to some extent, the present state. Once an action is applied, all relevant observations from sensors are gathered to derive the state of the system. Next actions are only applied once the state is assessed. To increase robustness, two concepts have been used: partial solutions allow to partially define a plan when relevant information is not yet available; flexible timelines assign time intervals (rather than fixed time points) to the start and end of the activities to make plans more flexible. Regarding expressiveness, HTN and timeline planning are the cornerstones of QuijoteExpress.

Combining different techniques from classical and applied planning is one of the key ideas of this thesis. Forward-chaining, planning enhancements or advanced heuristics from the classical world can help to boost the planner performance. Timeline planning and HTN can provide higher expressiveness levels while HTN also facilitates an easier inspection of the plans in a hierarchical manner.

With respect to execution, the study of the state-of-the-art revealed that none of the existing solutions was suitable for the two reference scenarios. It is important to remark that the number of executives is however very limited in comparison to the number of planners. Using SMACH would force the architecture to duplicate the definition of the domain and problem in DDL and the SMACH specific language which might cause several problems in terms of consistency, synchronization, etc. Furthermore, time is partially supported. With respect to Teer, it has a closer approach to timelines but it is rather limited in terms of temporal execution and conditional execution. T-Rex has been used with APSI planners in the GOAC experiment [30, 73, 36]. However, it imposes a whole architecture based on reactors that goes beyond the responsibilities of an executive. In addition, its reactor approach is highly redundant with the concept of ROS node, making its integration with many robotic platforms very difficult. The factors

affecting the evolution of planning and execution are different. The lack of standards and affordable platforms has a bigger impact in executives as they are closer to the hardware than planners. Moreover, there is no standard definition for what a plan is: even though timelines are fairly common, other approaches could be used. The irruption of ROS will certainly help to solve this situation.

Due to all these factors, a new timeline-based executive introduced in Section 5 has been developed. It is capable of managing flexible plans according to the formal definition provided by APSI.

3 Hierarchical Timeline Networks: A Novel Planning Formalism

Among the most relevant applied techniques, Hierarchical Task Networks or HTN [64] (see Section 2.3.2) and Timeline-based Planning or TLP [108, 71, 74] (Section 2.3.3) have been used in several systems and seem to complement each other. Even though the unification of these two formalisms could be relevant for multiple applications, little efforts have been conducted in this direction and clearly need further research. One exception is ASPEN (see Section 2.4.4), but its formalism has not been fully defined and it does not explore all the possibilities offered by the combination of both techniques.

This chapter introduces the Hierarchical Timeline Planning (HTLN [170]) formalism that aims to provide a solid definition of the main concepts that emerge from the combination of HTN and TLP. Along the chapter it will be demonstrated that HTLN is more expressive than HTN and TLP alone, improves the time and space complexity by reducing the O in the unfolding and scheduling phases while maintaining the soundness and completeness of TRF, increases plan robustness by means of sufficient planning and improves the planner performance by defining how to achieve parallel planning.

3.1 Background

3.1.1 Combining HTN and Temporal Planning Theories

Among other applications, TLP has been broadly used in the space sector (see Section 2.4.4). Its strongest point is its expressiveness power to represent the dynamics of the environment. On the other hand HTN is one of the most used planning techniques in real-world applications, in part because it allows to effectively encode knowledge into domain-independent planners. Moreover, the organization of tasks in hierarchies helps to simplify the modelling, which is one of the major problems that engineers need to face to deploy automated tools and at the same time produces better understandable plans.

Temporal and HTN planning are powerful techniques, but they have both some deficiencies that can be overcome by combining them in a new joint formalism that could satisfy the objectives described at the beginning of the chapter. HTLN [170] aims to unify HTN and TLP planning under a new formalism, trying to be as compliant as possible with their respective nomenclatures, even though some concepts would need to be added or redefined. While HTN theory is clearly defined, there is no “standard” TLP approach. HTLN is based on one specific TLP formalism called TRF [35], therefore there might be conceptual differences with other temporal formalisms.

HTLN is strictly more expressive than HTN because temporal relations are more expressive than temporal ordering and truth constraints while parametric relations are more expressive than variable binding relations. For example, in the conventional HTN representation[64] there is one single temporal relation \prec used to express precedence, such as $n \prec n'$ where $n, n' \in N$. The precedence relation cannot express all the possibilities of Allen’s relations such as $n \text{ DURING } n'$. Moreover, HTN does not allow to express linear functions between parameters. In consequence, there is a whole range of problems that can be represented in HTLN but not in HTN. Comparing other aspects such as performance becomes then irrelevant, as HTN is not capable of representing those problems relevant for HTLN.

More important is the comparison of HTLN and TLP as both of them are oriented to solve the same type of problems. The advantages of incorporating HTN to TLP are:

- Plan robustness: HTLN contributes to increase plan robustness in two different ways: (1) Allowing to define high level goals to be partially defined in situations of uncertainty (see Section 3.3.2); (2) Because the methods represent pieces of good plans that have been thoughtfully validated before being added to the Knowledge DataBase.
- Performance: As explained in Section 2.3.2 and more specifically in Section 3.3.3, HTN planners have shown great performance in part because the decomposition of complex goals allow to insert pieces of good plans (the methods) in the plan in just one step, helping to decrease the size of the search space.
- Expressiveness: Hierarchical decisions make HTLN more expressive than TLP as it is possible now to define complex and simple decisions and to express constraints over sets of partially-ordered decisions.
- Human factor: HTLN enables a more rational modelling and plan structure, closer to the organization of real systems and the way in which humans think. It also allows users to define preferences about the types of solutions they are willing to accept.
- Domain-dependent knowledge: Allows domain-independent planners to be easily customized for different problems just by replacing the set of HTN methods.

Section 3.3 will provide formal demonstrations for some of these properties based on the formalism presented in Section 3.2.

3.1.2 Hypergraph Theory

In mathematics, hypergraphs join together graph and set theories [132]. They have been applied in different areas such as the definition of complex data structures or optimization [11, 77]. In HTLN, they are used to represent the set of *decisions* and *n*-ary *relations* of the domain and problem. Besides their properties as data-structures, hypergraphs are useful for the mission planner because they are efficient computational structures to calculate parameters such as the shortest path between two decisions, connected components to divide a problem in subproblems, etc. In the following lines, the mathematical foundations are provided while in Chapter 4 it is analysed in depth how they are used in QuijoteExpress.

Hypergraph

A conventional hypergraph (represented in Figure 3.1) is a generalization of a graph whose edges, called hyperedges, can connect any number of nodes. Notice that a hyperedge is a set, and therefore all principles of set theory can be applied. HTLN uses however a nonconventional definition as it contains not only hyperedges but also hypernodes [132]. Both hypernodes and hyperedges can be considered themselves hypergraphs which is a key property in HTLN in order to represent hierarchical structures.

A hypergraph is formally defined as follows:

$$\begin{aligned}
 H &= (N, E) && \text{where:} \\
 N &= \{n_1, n_2, \dots, n_n\} && \text{finite set of hypernodes} \\
 E &= \{e_1, e_2, \dots, e_m\} && \text{finite set of hyperedges} \\
 \cup_{j=1}^m e_j &= N &&
 \end{aligned}
 \tag{3.1}$$

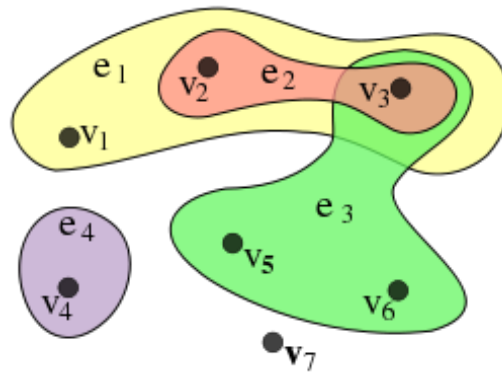


Figure 3.1.: Standard hypergraph representation without hypernodes (Courtesy of Wikipedia).

The domain and problem used as inputs of a HTLN-based planner can be translated to their underlying hypergraphs. These hypergraphs are used instead of the conventional data structures (sets of decisions and relations) by the pre-processor, some resolvers and heuristics as explained in Chapter 4.

Nodes and Hypernodes

A node is a mathematical abstraction used to represent an object (e.g. a decision in HTLN). A hypernode hn on the other hand represents a set of nodes and their associated hyperedges.

Hyperedges

Represent relations among a group of nodes or hypernodes (e.g. a temporal precedence among the nodes involved). Notice that, even though it has the same theoretical representation as hypernodes, a hyperedge has a completely different meaning which will be reflected in the way they are used in HTLN. While a hypernode represents a complex decision, a hyperedge represents a relation.

Properties

Hypergraphs are useful from a theoretical point of view in order to study different mathematical properties and from an applied point of view as an effective data structure to represent information. In consequence, HTLN uses hypergraphs to formalise its definitions and QuijoteExpress, a planner based on HTLN, uses them as data structures to represent decisions and relations. The hypergraphs used by both HTLN and QuijoteExpress are cyclic and directed.

The domain and problem definition might contain cycles depending on the properties of the problem at hand. For example, in the FASTER domain presented in Section 6.3.3, the traverse behaviour in charge of moving the rover from point A to B contains a cycle that is repeated until the rover reaches B.

In TRF, temporal relations are directed while parameter relations are not. In case the internal hypergraph representation does not allow to combine directed and undirected edges, it is possible to translate an (undirected) not instantiated parametric relation such as f_{param} involving m values v_1, v_2, \dots, v_m

into a number of directed relations E_f by calculating the 2-permutation of its values. The number of edges produced is $|E_f| = \frac{m!}{(m-2)!}$. Formally:

$$f_{param_o}(v_1, v_2, \dots, v_m) \Rightarrow E_f = f_{param_1}, \dots, f_{param_m}, f_{param_i} = v_i, v_j / v_i, v_j \in f_{param_o}, v_i \neq v_j \quad (3.2)$$

3.2 HTLN Formalism

HTLN provides formal definitions of the building blocks required for hierarchical timeline planning, leaving aside on purpose any specific implementation detail, which will be covered in Chapter 4. Table 3.1 summarizes the main concepts of TLP, HTN, and their equivalences in HTLN.

HTN	TRF	HTLN	Symbol
Primitive task	Value: state variable or resource	Primitive state variable value or resource value	$v^p \in V^p$
Compound or Complex task	-	Complex state variable value	$v^c \in V^c$
Action (Instantiated task)	Decision: ValueChoice or Activity	Decision: Simple/complex ValueChoice or Activity	d, d_{v^p}, d_{v^c}
Method	-	Method	$m \in M$
Task Network	Decision Network	Decision Network	dn
Decomposition	-	Decomposition	δ
Transition function	Transition function	Transition function	γ

Table 3.1.: Combining HTN and APSI nomenclature in HTLN.

3.2.1 Values and Decisions

In HTN, a **task** is defined as $t(p_1, \dots, p_n)$ where t is a symbol and p_i is a term. If t is an operator, then the task is **primitive**, otherwise the task is **complex**¹. TRF tasks are represented by means of values, formally $v = \langle s_v \in S, T_v = \langle t_{v_1}, \dots, t_{v_m} \rangle, \tau_v = [t_s, t_e] \rangle$, where the symbol s_v is the name of the value, T_v is the set of parameters of v and τ_v the minimal and maximal allowed temporal duration of the value. TRF has two type of values, state variable values (e.g. *going_to(integer, float, float)*) and resource values (e.g. *battery.REQUIREMENT(float)*) but it does not distinguish between primitive and complex values. The equivalent in TRF of a HTN **action** (instantiated task) is a **decision** represented as a pair $\langle time - frame, value \rangle$ (see Section 2.4.4) where the time-frame is the time in which the value will actually happen according to the plan. TRF has two types of decisions, one for each type of value: ValueChoice (VC) for state variable components and Activity for resources.

¹ The conventional name in HTN nomenclature is compound. Along the thesis, the word *complex* will be used instead, as it is considered by the author to be more representative.

HTLN increases the expressiveness of TRF by adding the concept of complex state variable value, equivalent to the concept of complex task in HTN. The set of all primitive and complex values in a domain are represented as V^p and V^c respectively, where:

$$\begin{aligned} V^p \cap V^c &= \emptyset \\ V^{all} &= V^p \cup V^c \end{aligned} \tag{3.3}$$

It is very easy to infer the complexity property of a value v^c from the domain just by analysing whether v^c is reference of any method in the domain (complex) or not (primitive). On the other hand, HTLN inherits “as is” the definition of resource value from TRF because there is no straightforward utility for the concept of complex activity as it happens with complex ValueChoices and it would increase the complexity to calculate resource consumption. Without loss of generality, value and decision will be used from now on without specifying the type of component (state variable or resource) whenever the type can be unambiguously determined from the context.

3.2.2 Decision Network (dn)

HTN and TRF represent problems as partially supported networks called **task network** and **decision network** (dn) respectively. Both are modelled as a graph or hypergraph $w = (N, E)$ containing a set of tasks (HTN) or decisions (TRF) which are the nodes of the hypergraph and a set of relations among the nodes which are the edges/hyperedges of the hypergraph. In HTLN, complex decisions are represented as hypernodes that might also contain other subhypernodes. A dn is represented in HTLN as a graph of hypergraphs, formally:

$$dn = (v^c, N, E, dn^{sub}) \tag{3.4}$$

where v^c is the optional value of the network when it is considered as a single value, N is the list of nodes, E the list of edges and dn^{sub} the list of subhypernodes where each subhypernode is itself a dn .

When a dn is instantiated, every value v in the network (including v^c and those in N) is transformed in a node containing a decision with value v . Every element in the network (decision, parameter or temporal element) is assigned a unique identifier to distinguish it unambiguously.

3.2.3 Primitive Value (v^p)

The representation of a primitive value is the same as in TRF. However, its definition need to be extended to cope with HTN. A primitive value (v^p) cannot be further decomposed and is directly executable. It is instantiated in HTLN as a decision (d_{v^p}), that is a node of the underlying hypergraph.

Example 5 *In the FASTER scenario, TraversePrimary is a primitive value in charge of moving the primary rover. It has a boolean parameter that indicates whether the destination is actually a waypoint in the path (a subtarget). Because the specific coordinates do not play any role in the generation of the plan, they are hidden in the model. The lower bound for its*

execution is five seconds and the upper ten minutes. *TraversePrimary* is formally defined as:

TraversePrimary(*bool?isWP*)[5000, 600000];

Formally:

$v = \langle pred_v, lb_v, ub_v \rangle;$

$pred_v = \langle Name = TraversePrimary, T_v = \langle \langle ?isWP, FALSE, TRUE \rangle \rangle \rangle;$

$lb = 5000;$

$ub = 600000;$

3.2.4 Complex Value (v^c)

Unlike HTN, the instantiation d_{vc} of a complex value can be represented in two different ways depending on its decomposition status: (1) In case it has not yet been decomposed, d_{vc} is represented by a Value-Choice decision (a node) that cannot be executed; (2) In case it has been decomposed, d_{vc} is represented as a decision network (a hypernode), abbreviated as $dn_{d_{vc}}$ or simply dn_d (see the tree below).

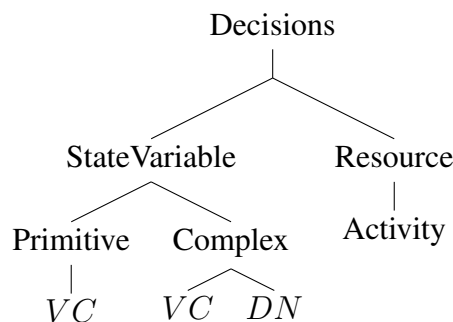
Moreover, a decomposed decision represented by the network dn_d , can be used in two ways:

- Single decision (hypernode): The planner uses dn_d as if it were a single decision (the value of which is d_{vc}) in case it wants to make modifications that affect dn_d as a whole. The network itself is seen as a black box.
- Decision Network (hypergraph): The planner can use dn_d as a decision network (a subproblem) in case it wants to make modifications on dn_d decisions or relations.

The duality in the representation of complex decisions and in the interpretation of decision networks is a powerful novelty with respect to ASPEN (the only other timeline-based planner with HTN capabilities) and with any other HTN planner to the best of our knowledge. The main benefits of this approach, theoretically demonstrated in Section 3.3, are:

- Better planning performance: Interpreting dn_d as a single decision d_{vc} , modifications on d_{vc} directly affect the whole subnetwork.
- Handle uncertainty: In case it is unknown how to achieve a complex goal d^c during planning time, d^c is modelled as a cd . During execution, once the required information is available, a decomposition method is chosen for d^c and the plan is completed.
- More intuitive results: Plans are organized in multiple layers of increasing detail/complexity.
- Higher expressiveness than HTN and TRF.

Unlike HTN, the concept of executability and valid plan are decoupled in HTLN as explained in Section 3.3.2. It is not mandatory to decompose a complex decision to generate a valid plan, but it is to generate an executable one.



Example 6 In the *FASTER* scenario, *TraverseCycle* is a complex value used to move a robot along a segment from the initial to the destination position. *TraverseCycle* can be decomposed in two methods, *TraverseCycle – PR* and *TraverseCycle – Team*, defining a sequence of activities repeated in a loop until the robot reaches the destination position. *TraverseCycle* has a boolean parameter that indicates whether the traverse is done in cooperative mode between the primary and scout rovers or just by the primary. Similarly as in the previous example, the specific coordinate variables are ignored. The lower bound for its execution is two seconds and the upper 20 minutes. Its formal definition has the same format as a primitive value, with the only difference being that *TraverseCycle* has associated one or more decomposition methods:

TraverseCycle(bool?*inTeam*)[2000, 1200000];

Formally:

$v = \langle pred_v, lb_v, ub_v \rangle;$

$pred_v = \langle Name = TraverseCycle, T_v = \langle (?inTeam, FALSE, TRUE) \rangle \rangle;$

$lb = 2000;$

$ub = 1200000;$

3.2.5 Domain (D)

In HTN, a domain is defined by a set of operators and methods $D = (O, M)$. On the other hand, a TRF domain is constituted by a set of components and a domain theory that states the relations among the values of the components. HTLN adds to TRF's definition a list of methods in order to support HTN. In consequence, the decisions of a domain in HTLN are organized in a tree structure with $k + 1$ levels of abstraction, where level k corresponds to the root of the tree with the highest level of abstraction while level 0 corresponds to the leaves containing the most detailed components. A complex value v^c might have associated several methods (see Section 3.2.7) where more than one could be applicable for a given problem P representing a branching point for the decomposer resolver.

3.2.6 Problem (P)

In HTN, a problem is defined as $P = (s_0, w, D)$, where s_0 is the initial state represented as a list of atoms, w the initial task network containing the tasks to achieve and D the domain with a list of operators and methods. A problem in TRF is represented as $P = (st_{init}, D)$ where st_{init} is the initial state defined as a partially supported decision network that contains a list of pairs $\langle TL_i, d_i \rangle$ with the initial value for each timeline and the goals for some timelines, and D is the domain.

HTLN takes the approach of classical planning where the initial state st_{init} contains the network dn_{init} with the initial value of all the timelines while the goal state st_{goal} contains the network dn_{goal} with the list of goals. While dn_{init} must be fully defined, that is, the value of all the components must be specified at $t = 0$, dn_{goal} might not specify the value for some components. This fact plays an important role in the type of search strategy (forward or backward chaining) used in QuijoteExpress as explained in Section 4.3.

As the initial and goal networks dn_{init} , dn_{goal} do not decompose any higher level decision, they cannot behave as simple nodes. Therefore a problem network dn_x is always represented as a hypergraph with an internal value v_x equals to null. The Table 3.2 presents a correlation between the elements of the hypergraph and the elements of HTLN.

Element	Description
Node	Primitive d_{vp} or not-decomposed complex d_{vc} decision.
Hypernode	Decomposed complex decision d_{vc} .
Hyperedge	Constraint (Parameter, Temporal) between a set of nodes.
Hypergraph	Problem or plan.

Table 3.2.: Correlation between hypergraph and HTLN elements.

3.2.7 Method (m)

A **method** is a HTN mechanism employed by the user to express control-knowledge that tells the planner which areas of the search space should be explored by telling the only valid ways in which complex values can be decomposed. The set of methods contains mission-dependent information that can be modified for the same domain to achieve different purposes.

A method is formally defined as $m = (name_m, v_m^{ref}, dn_m^{dec})^2$ where $name_m$ is a unique name for the method, v_m^{ref} the complex value to be decomposed, called **reference** and $dn_m^{dec} = (v_m = v_m^{ref}, N_m, E_m, dn^{sub} = \emptyset)$ is a decision network called the **target network** that decomposes the reference in smaller, more detailed decisions.

Regarding v_m , contrary to problem networks which can be exclusively used as decision networks, all target networks can also be used as simple decisions, the reason why dn_m^{dec} has an associated value v_m corresponding to the reference value that the network decomposes. This feature allows HTLN to create relations involving the complex value even when it has been already decomposed, affecting the target network as a whole.

Focusing on the target network dn_m^{dec} , besides the internal relations among its decisions, it requires a number of additional temporal and parameter relations between the complex decision v_m and the rest of decisions.

In case dn_m^{dec} is totally ordered, two additional temporal relations $f_{temp}^{before}(v_m, first(N_m))$ and $f_{temp}^{after}(v_m, last(N_m))$ must be added, where $first$ returns the first decision of the network according to the temporal order and $last$ the last one. Otherwise, for each decision $d_{m_i} \in N_m$, a relation $f_{temp}^{contains}(v_m, d_{m_i})$ is added, meaning that each decision in dn_m^{dec} must be executed within the time frame assigned to v_m , which will be equal once v_m is instantiated to the time frame of d_{vc} , the complex decision that will be decomposed in dn_m^{dec} .

Moreover, for any parameter $t_{v_i^{ref}}$ in v_m^{ref} unifiable (see Section 3.2.11) with another parameter $t_{d_{m_i}}$ in a decision $d_m \in N_m$, an equal constraint $f_{param}^{equal}(t_{v_i^{ref}}, t_{d_{m_i}})$ must be added.

In case these relations are not specified in the method, the process to add them comes with no cost at planning time because they will be added during the pre-processing phase.

With respect to the subnetworks, dn^{sub} will be originally empty in every method as no complex values in dn_m^{dec} is decomposed yet.

The set of methods M of a domain constitutes the domain-dependent information used by a domain-independent planner to limit the size of the search space. If there is at least one complex value v^c in the domain that cannot be decomposed in primitive values, then the *model is not complete* and it is not possible to generate executable plans for problems containing v^c .

² A more relaxed notation $m = (v_m, dn_m)$ will be occasionally used for the sake of simplicity.

Example 7 The following example shows a formal description of two methods relevant for the complex decision *Traverse* of the *FASTER* domain:

```

name : m_traverse_nominal ;
id : tn ;
reference : Traverse(graph, p_init, p_dest) ;
target :
  vtn = Traverse(graph0, p_init0, p_dest0) ;
  Ntn = {ntn1 : InitTravGraph(graph1), ntn2 : FindRoute(p_init2, p_dest2), ntn3 :
    TraverseCycle(path3)} ;
  Etn = {etn1 : ftempbefore(ntn1, ntn2), etn2 : ftempbefore(ntn2, ntn3), etn3 : ftempbefore(vtn, ntn1), etn4 :
    ftempafter(vtn, ntn3), etn5 : fparamequal(graph0, graph1), etn6 : fparamequal(p_init0, p_init2), etn7 :
    fparamequal(p_dest0, p_dest2)} ;
  dntndec = ∅ ;

```

```

name : m_traverse_non_nominal ;
id : tnn ;
reference : Traverse(graph, p_init, p_dest) ;
target :
  vtnn = Traverse(graph0, p_init0, p_dest0) ;
  Ntnn = {ntnn1 : BringBackSR(p_init1, p_dest1)} ;
  Etnn = ∅ ;
  dntnndec = ∅ ;

```

The method *m_traverse_nominal* decomposes the complex decision *Traverse* in a network which contains three nodes and seven relations. The relations can be classified into three subgroups (see Section 3.2.12):

- {e_{tn₁}, e_{tn₂}}: Ordering relations between the subdecisions.
- {e_{tn₃}, e_{tn₄}}: Extra temporal relations between v^{ref} and the subdecisions in dn^{dec}.
- {e_{tn₅}, e_{tn₆}, e_{tn₇}}: Extra parameter relations between v^{ref} and the parameters of the subdecisions in dn^{dec}.

In order to guarantee a good performance during planning (see Section 3.3.3), the terms (parameters or decisions) are assigned specific levels of abstraction in such a way that a term in a given level does not appear anywhere in higher levels of abstraction. In other words, the number of terms monotonically increases with the level of detail.

3.2.8 Relations (f)

TRF has two types of relations (Section 2.4.4): (1) Unary/binary temporal (instant and interval) constraints such as $t_1 \text{ BEFORE } t_2$ where *BEFORE* is an Allen's relation; (2) n-ary equality/inequality $v \neq v'$ or linear parametric relations such as $x + 2y = k^3$.

On the other hand HTN typically has three types (Section 2.3.2): (1) Variable binding such as $v = v'$; (2) Temporal ordering such as $n \prec n'$; (3) Truth constraints such as (n, l) which means that the literal l must be true immediately after the task n .

TRF constraints represent a superset of HTN, therefore HTLN does not extend the definition of relation from TRF.

3.2.9 Resolver (ρ)

Some planners such as ASPEN or AP² have a set \mathbb{P} of resolvers used iteratively to refine a problem until a solution is found or it is demonstrated that none exists. Designing a planner based on resolvers is an architectural decision suitable when the planning process can be clearly divided in different steps, but it does not enforce any specific type of planning such as flaw-resolution or forward-chaining.

A **resolver** $\rho \in \mathbb{P}$ is a procedure intended to fix some specific type of flaw. In TRF, it is formally defined as $\rho(D, dn_i, dn_{goal}) = \{d^+, d^-, f^+, f^-\}$ where D is a formal description of the domain, dn_i the current problem, dn_{goal} the goal network and d^+, d^-, f^+, f^- the lists of decisions (ValueChoice and/or Activity) and relations (temporal and/or parametric) to be added/retracted respectively. Given a network dn_i , the resolver tries to fix it by adding/retracting decisions ($\{d^+, d^-\}$) and relations ($\{f^+, f^-\}$) in order to achieve dn_{goal} .

Different types of planning paradigms require different solving steps. In state-space, the equivalent of a resolver is the algorithm in charge of searching all applicable actions for a given state. Plan-space partial order planning (POP) is oriented towards the collection and resolution of flaws, divided in two main steps: (1) Unfolding: Adds supporting decisions for the goals in the network; (2) Scheduling: Add relations based on causal links that temporally order the new activities added during the unfolding step. In HTN, resolvers might vary but generally extend partial order planning with one additional step known as task reduction or decomposition.

TRF presents some similarities with respect to POP as it is also oriented to the resolution of flaws. While the unfolding process is quite similar, scheduling is a more generic/powerful variant of the causal links based on Allen's temporal relations. Moreover, it adds an additional one: (3) Timeline Completion, which fills the gaps of the timelines where no state is specified. HTLN adds to these steps the **decision decomposition**.

The order in which each resolver should be applied depends on the specific implementation and would be typically heuristically determined

3.2.10 Transition Function (γ)

In classical and HTN planning, the **state-transition function** γ is used to calculate the new state s' of the problem by applying the effects of a given action a to the current state s , formally $\gamma(s, a) = s'$. In

³ The terms relation and constraint will be indistinctly used along the text as HTLN relations always impose constraints in the domain of the variables involved.

HTLN, the state is defined by a decision network rather than a set of values and instead of an action a , the output of one or several resolvers is used.

Given the problem network $dn_i = (v_i, N_i, E_i, dn_i^{sub})$ and $\{\rho\}$ a list of resolvers, suppose without loss of generality that $\{\rho\}$ is formed by just one resolver $\{\rho_x\}$. The transition function γ under the application of the resolver ρ_x is calculated as shown in equation 3.5.

$$\begin{aligned} \gamma(dn_i, \rho_x) = dn_{i+1} = & (v_{i+1}, N_{i+1}, E_{i+1}, dn_{i+1}^{sub}) \\ & v_{i+1} = null \\ & N_{i+1} = N_i \setminus d^- \cup d^+ \\ & E_{i+1} = E_i \setminus f^- \cup f^+ \\ & dn_{i+1}^{sub} = dn_i^{sub} \setminus d^- \in dn_i^{sub} \cup d^+ \in DN \end{aligned} \quad (3.5)$$

Intuitively, the list of nodes N_{i+1} and edges E_{i+1} are calculated from their predecessors in dn_i by adding and deleting the corresponding lists generated by the resolver. Regarding the list of subnetworks, dn_{i+1}^{sub} adds all the decisions in d^+ which are decision networks and removes those decisions from d^- that were subnetworks in dn_i .

Prior to the description of the decomposition process, it is required to formally introduce the concept of unification (used by the decomposition) and the necessary conditions for unification.

3.2.11 Unification

Unification attempts to identify and delete duplicated elements from a decision network. The process is complex and requires a precise definition.

Definition 43 (Constraints for Unification) *Two variables of the same type can be unified when there is a not null set of solutions for which the two variables can share the same value.*

In practice, checking the bounds of two variables resulting after the propagation of the posted constraints is just a necessary but not sufficient condition to check the unifiability of the two variables because propagation will give information about the values that are not contained in any solution but no information about the values that will be in the solution. To check unifiability, it would be necessary to post an equality constraint among the variables and check that a solution exists to be sure that they can unify. What is possible to check is if two variables cannot unify. To state a *necessary* condition for unification, it is enough to check if there is an empty intersection between the domains of the two variables. The condition can be sufficient if the two data variables are constants or if one of the two variables is not constrained, in which case it can always unify with another variable [139].

In those cases where the unification conditions are not sufficient, a TRF/HTLN planner can only create a branch in which the two elements are unified and another in which they are not. As search continues, one of the branches will be proved to be inconsistent because some constraints could not be satisfied.

Given a decision d and another element e (decision, parameter or relation) with which d can be unified, there exists a most general unifier θ which application produces a result different from \emptyset :

$$CanUnify(d, e) = \top \Rightarrow \exists \theta, \theta(d, e) = \theta e, \theta e \neq \emptyset \quad (3.6)$$

Unification is different depending on the type of e . Therefore, it will be iteratively explained in increasing level of complexity of the element type.

Parameter Unification

The necessary conditions to unify two parameters can be formally expressed as:

$$t_v, t_{v'} \text{ terms} | \text{type}(t_v) = \text{type}(t_{v'}) \wedge D(t_v) \cap D(t_{v'}) \neq \emptyset \Rightarrow \text{CanUnify}(t_v, t_{v'}) = \top \quad (3.7)$$

Equation 3.7 indicates that the parameters $t_v, t_{v'}$ can be unified if both share the same type and the intersection of their domains is not empty. In case the two parameters have the same type and they are not instantiated (the parameter's domain is not restricted by any constraint), there is no need to compare the domains of the parameters as they would be always equal.

The unification under θ proceeds as follows:

$$t_v, t_{v'} \text{ terms} | \text{CanUnify}(t_v, t_{v'}) = \top \Rightarrow \theta(t_v, t_{v'}) = \theta t_{v'} | D_{\theta t_{v'}} \leftarrow D_{t_v}, \text{value}_{\theta t_{v'}} \leftarrow \text{value}_{t_v} \quad (3.8)$$

In case t_v and $t_{v'}$ are unifiable, then $t_{v'}$ will be assigned the domain and specific value from t_v .

Decision Unification

The necessary conditions to unify two decisions is slightly more complicated:

$$d_v, d'_{v'} \text{ decisions} | s_v = s_{v'} \wedge |T_v| = |T_{v'}| \wedge \forall t_{v_i} \in T_v \exists t'_{v'_i} \in T_{v'}, \text{CanUnify}(t_{v_i}, t'_{v'_i}) = \top \\ \Rightarrow \text{CanUnify}(d_v, d'_{v'}) = \top \quad (3.9)$$

Given two values d_v and $d'_{v'}$, they can be unified in case their value symbols are equal, they share the same number of parameters and each pair of parameters $(t_{v_i}, t'_{v'_i})$ can be unified.

If d_v and $d'_{v'}$ are unifiable, the unification under θ proceeds as follows:

$$d_v, d'_{v'} \text{ decisions} | \text{CanUnify}(d_v, d'_{v'}) = \top \Rightarrow \theta(d_v, d'_{v'}) = \theta d'_{v'} | \\ \tau_{\theta d'_{v'}} \leftarrow \tau_{d_v} \\ \theta(t_{d_{v_i}}, t'_{d'_{v'_i}}) \forall i \in |T_{d_v}| \quad (3.10)$$

The unification of d_v and $d'_{v'}$ produces a decision $\theta d'_{v'}$, the time frame of which is the one of d_v and all its parameters are the result of the unification of the parameters of d_v and $d'_{v'}$. Notice that the definition of unification for two nodes of a decision network can be directly extracted from the previous definition as follows:

$$n_d, n_{d'} \text{ nodes} | \text{CanUnify}(d, d') = \top \Rightarrow \text{CanUnify}(n_d, n_{d'}) = \top \quad (3.11)$$

The unification of two nodes $n_d, n_{d'}$ produces a new node $\theta n_{\theta d'}$ which decision is the result of unifying the decisions of the two nodes. Formally:

$$n_d, n_{d'} \text{ nodes} | \text{CanUnify}(d, d') = \top \Rightarrow \theta(n_d, n_{d'}) = \theta n_{\theta d'} | \\ \theta(d, d') = \theta d' \quad (3.12)$$

Unification with a Relation

The necessary condition to unify a value d with a relation f_{type} depends on the type of relation.

In the case of a temporal relation f_{temp} , the necessary condition to unify it with d is:

$$\begin{aligned} d \text{ decision, } f_{temp} \text{ relation} &| \exists d_f \in \text{scope}(f_{temp}) \wedge \text{CanUnify}(d, d_f) = \top \\ &\Rightarrow \text{CanUnify}(d, f_{temp}) = \top \end{aligned} \quad (3.13)$$

If d can be unified with any of the decisions in the scope of f_{temp} then d and f_{temp} are unifiable as follows:

$$\begin{aligned} d \text{ decision, } f_{temp} \text{ relation} &| \text{CanUnify}(d, f_{temp}) = \top \\ \Rightarrow \theta(d, f_{temp}) &= \theta f_{temp} | \text{scope}(\theta f_{temp}) = \text{scope}(f_{temp}) \setminus d_f \cup d \end{aligned} \quad (3.14)$$

The unification of d and f_{temp} produces θf_{temp} , a relation equal to f_{temp} in which scope d_f is replaced by d .

In case of a parameter relation f_{param} , the necessary condition to unify it with d is:

$$\begin{aligned} d \text{ decision, } f_{param} \text{ relation} &| \exists t_d \in T_d, \exists t_f \in \text{scope}(f_{param}) \wedge \text{CanUnify}(t_d, t_f) \\ &\Rightarrow \text{CanUnify}(d, f_{param}) = \top \end{aligned} \quad (3.15)$$

If the parameter t_d of the decision d can be unified with one parameter t_f in the scope of the relation f_{param} , then d and f_{param} can be unified as follow:

$$\begin{aligned} d \text{ decision, } f_{param} \text{ relation} &| \text{CanUnify}(d, f_{param}) = \top \\ \Rightarrow \theta(d, f_{param}) &= \theta f_{param} | \text{scope}(\theta f_{param}) = \text{scope}(f_{param}) \setminus t_f \cup t_d \end{aligned} \quad (3.16)$$

Which is very similar to the unification for a temporal relation.

Unification with a Decision Network

Finally, the unification of a value d with a decision network dn proceeds by iteratively applying the unification to each of the decisions and relations of the network (in case the corresponding necessary conditions are met), to each subnetwork and to the network's value. Formally:

$$\begin{aligned} \theta(d, dn_m = (v_m, N_m, E_m, dn_m^{sub})) &= \\ \theta(d, v_m) \theta(d, d_m) \forall d_m \in N_m &| \text{CanUnify}(d, d_m) = \top, \\ \theta(d, f_m) \forall f_m \in E_m &| \text{CanUnify}(d, f_m) = \top, \\ \theta(d, dn_m^{sub} _i) \forall dn_m^{sub} _i \in &dn_m^{sub}, \end{aligned} \quad (3.17)$$

Notice that during decomposition the set of subnetworks dn^{sub} is empty, therefore no propagation of the unification is required.

3.2.12 Decomposition Resolver (ρ_δ)

Given a problem network $dn_i = (v_i = null, N_i, E_i, dn_i^{sub})$ and a complex decision $d_{vc} \in N_i, d_{vc} = \langle \tau_d, v \rangle$, being v a state variable value with a predicate $pred_v = \langle s_v \in S, T_v = \langle t_{v_1}, \dots, t_{v_m} \rangle \rangle$. Let $m = (v_m^{ref}, dn_m^{dec})$ where $dn_m^{dec} = (v_m, N_m, E_m, dn_m^{sub})$ is a relevant, applicable method for v and θ the most general unifier of d_{vc} and dn_m^{dec} . The decomposition of d_{vc} requires a sequence of activities explained in the following subsections.

Selection

The only methods interesting to decompose the complex decision d_{vc} are those that are both applicable in the current state s and relevant for d_{vc} .

In HTN, applicability is defined as follows:

Definition 44 (Applicability) *A method m is applicable in a state s if $precond^+(m) \subseteq s$ and $precond^-(m) \cap s = \emptyset$.*

This definition is not valid for HTLN. Even though the concept of *precondition*, which does not exist in TRF, can be easily extended to relation, proving the satisfiability condition of the relations in m 's target network requires to actually solve the problem. The necessary but not sufficient condition is that a method could be applicable if none of its relations cannot be satisfied in s , in other words, if there is a relation in m that would be violated in s , then the method is not applicable. Formally:

$$\forall f \in E_m, \text{Applicable}(f, dn_i) \neq \perp \Rightarrow \text{Applicable}(dn_m^{dec}, dn) = 1/2^4 \quad (3.18)$$

The specific conditions to be checked by $\text{Applicable}(f, dn_i)$ depend on the implementation, in this case on APSI, and are out of the scope of this chapter.

On the other hand, the definition of relevance in HTN can be easily adapted to HTLN.

Definition 45 (Relevance) *A method m is relevant for a complex decision d_{vc} in case v^c can be unified with v_m^{ref} . Formally:*

$$\forall m \in M, m = \langle v_m^{ref}, dn_m^{dec} \rangle | \text{CanUnify}(d_{vc}, v_m^{ref}) = \top \Rightarrow m \in M_{vc}^{select} \quad (3.19)$$

The selection step provides a list of methods proven not to be not applicable in dn_i and relevant for d_{vc} . In theory, each method $m \in M_{vc}^{select}$ represents a new branch in the search space created as a result of the application of the decomposition to the current state s . Specific details about the order in which the children will be chosen for expansion depends on the specific implementation and will be covered in Chapter 4.

⁴ 1/2 means Maybe in three-valued logic.

Instantiation

Once a method is selected, an instantiation $dn_d = (n_d, N_d, E_d, dn_d^{sub})$ of the target network dn_m^{dec} is created. As m is relevant for d_{vc} , there exists a unifier θ between d_{vc} and dn_m^{dec} ⁵, which application to a decision, described by Equation 3.10, is adapted here to the case of a network's decision as follows:

$$\theta(d_{vc}, n_d) = \theta n_d | \tau_{\theta n_d} \leftarrow \tau_{d_{vc}}, \nu_{\theta n_d} \leftarrow \nu_{d_{vc}} \quad (3.20)$$

The instantiated target network θdn_d is also a decision which gets the time frame and value (symbol, list of parameters and for each parameter its specific value) from d_{vc} .

Decomposition

Once a relevant method for d_{vc} has been selected and instantiated, it is possible to proceed to the decomposition. To do so, all relations in E_d (the instantiated target network) and E_i (the problem network) unifiable with d_{vc} are unified according to equations 3.14 and 3.16, formally:

$$\forall f \in \{E_i, E_d\} | CanUnify(d_{vc}, f) = \top \Rightarrow \theta(d_{vc}, f) = \theta f \in \theta F \quad (3.21)$$

where F is the set of all relations that can be unified with d_{vc} and θF is the set of unified relations.

The result of the decomposition of $d_{vc} \in N_i$ by dn_d under the most general unifier θ is formally expressed as:

$$\begin{aligned} \rho_\delta(dn_i, d_{vc}, dn_d, \theta) &= \{d^+, d^-, f^+, f^-\} \\ d^+ &= \{dn_d, N_d\} \\ d^- &= d_{vc} \\ f^+ &= \theta F \\ f^- &= F \end{aligned} \quad (3.22)$$

The nodes N_d plus the network dn_d itself are the list of decisions to be added while d_{vc} has to be deleted. With respect to the relations, all the unifiable relations F must be retracted and the resulting unified relations θF added. The output of ρ_δ is used to create an evolution of the problem dn_i as shown in Equation 3.5.

Example 8 *Continuing with the FASTER rover domain, the example below contains two activities: (1) Initial traverse with the two rovers towards a target; (2) Once the destination is reached, establish communication.*

```

name : dn0 ;
id : 0 ;
n0 = null ;
N0 = {n1 : ⟨τd1 = [0, 10], νd1 = Traverse(graph1, pinit1, pdest1)⟩ ; n2 :
  ⟨τd2 = [11, 20], νd2 = Communicate(file2)⟩} ;
E0 = {e1 : ftempbefore(n1, n2)} ;
dn0sub = {∅} ;

```

⁵ Recall that $\nu_m = \nu_m^{ref}$ is the value related to the target network.

Suppose that the next resolver chosen to be applied to the problem dn_0 is the decomposer over the decision d_1 in the node n_1 . The following lines show the steps required to decompose the *Traverse* goal in subgoals.

First, it is required to find all applicable and relevant methods for $Traverse(graph, p_{init}, p_{dest})$. Previously, two of them, named $m_traverse_nominal$ and $m_traverse_non_nominal$ were presented:

$$\begin{aligned} CanUnify(d_1, v_{tn}^{ref}) &= \top \wedge Applicable(dn_{tn}, dn_0) = 1/2 \\ CanUnify(d_1, v_{tnn}^{ref}) &= \top \wedge Applicable(dn_{tnn}, dn_0) = 1/2 \end{aligned} \quad (3.23)$$

Next, one of these methods must be heuristically selected. Suppose without loss of generality that the first one is chosen. An instantiation of the network is created, assigning a unique *id* to the network, nodes and relations:

```

name : m_traverse_nominal ;
id : a ;
reference : Traverse(graph, p_init, p_dest) ;
target :
  n_a0 = Traverse(graph_a0, p_init_a0, p_dest_a0) ;
  N_a = {n_a1 : InitTravGraph(graph_a1); n_a2 : FindRoute(p_init_a2, p_dest_a2); n_a3 :
    TraverseCycle(path_a3)} ;
  E_a = {e_a1 : f_temp^before(n_a1, n_a2); e_a2 : f_temp^before(n_a2, n_a3); e_a3 : f_temp^before(n_a0, n_a1); e_a4 :
    f_temp^after(n_a0, n_a3); e_a5 : f_param^equal(graph_a0, graph_a1); e_a6 : f_param^equal(p_init_a0, p_init_a2); e_a7 :
    f_param^equal(p_dest_a0, p_dest_a2)} ;
  dn_a^dec = ∅ ;

```

The unifier θ between the node to be decomposed, d_1 and the instantiation dn_a of $m_traverse_nominal$ is:

$$\theta(d_1, dn_a) = (n_3 \leftarrow n_{a0}; n_4 \leftarrow n_{a1}; n_5 \leftarrow n_{a2}; n_6 \leftarrow n_{a3}; e'_1 \leftarrow e_1; e_2 \leftarrow e_{a1}; e_3 \leftarrow e_{a2}; e_4 \leftarrow e_{a3}; e_5 \leftarrow e_{a4}; e_6 \leftarrow e_{a5}; e_7 \leftarrow e_{a6}; e_8 \leftarrow e_{a7}; graph_3 \leftarrow graph_{a0}; p_{init_3} \leftarrow p_{init_{a0}}; p_{dest_3} \leftarrow p_{dest_{a0}}; graph_4 \leftarrow graph_{a1}; p_{init_5} \leftarrow p_{init_{a2}}; p_{dest_5} \leftarrow p_{dest_{a2}}; path_6 \leftarrow path_{a3})$$

The unification of d_1 (the decision in n_1) and d_{a0} (the decision that represents the target network as a whole) according to equation 3.17 by means of θ is:

$$\theta(d_1, d_{a0}) = d_3 | \tau_{d_3} \leftarrow [0, 10], v_{d_3} \leftarrow Traverse(graph_3, p_{init_3}, p_{dest_3})$$

while the unification with the rest of the decisions in dn_a is:

$$\begin{aligned} \theta(d_1, d_{a1}) &= d_4 | \tau_{d_4} \text{ not grounded}, v_{d_4} \leftarrow InitTravGraph(graph_4) ; \\ \theta(d_1, d_{a2}) &= d_5 | \tau_{d_5} \text{ not grounded}, v_{d_5} \leftarrow FindRoute(p_{init_5}, p_{dest_5}) ; \\ \theta(d_1, d_{a3}) &= d_6 | \tau_{d_6} \text{ not grounded}, v_{d_6} \leftarrow TraverseCycle(path_6) ; \end{aligned}$$

Next, all the relations in dn_0 and θdn_a unifiable with d_1 are unified according to equations 3.14 and 3.16. The resulting relations are:

$$\begin{aligned}\theta(d_1, e_1) &= e'_1 : f_{temp}^{before}(n_3, n_2); \\ \theta(d_1, e_{a1}) &= e_2 : f_{temp}^{before}(n_4, n_5); \\ \theta(d_1, e_{a2}) &= e_3 : f_{temp}^{before}(n_5, n_6); \\ \theta(d_1, e_{a3}) &= e_4 : f_{temp}^{before}(n_3, n_4); \\ \theta(d_1, e_{a4}) &= e_5 : f_{temp}^{after}(n_3, n_6); \\ \theta(d_1, e_{a5}) &= e_6 : f_{param}^{equal}(graph_3, graph_4); \\ \theta(d_1, e_{a6}) &= e_7 : f_{param}^{equal}(p_{init_3}, p_{init_5}); \\ \theta(d_1, e_{a7}) &= e_8 : f_{param}^{equal}(p_{dest_3}, p_{dest_5});\end{aligned}$$

The output of the decomposer using equation 3.22 is:

$$\begin{aligned}\rho_\delta(dn_0, n_1, m_traverse_nominal, \theta) &= \\ d^+ &= \{n_3; n_4; n_5; n_6\}; \\ d^- &= \{n_1\}; \\ f^+ &= \{e'_1; e_2; e_3; e_4; e_5; e_6; e_7; e_8\}; \\ f^- &= \{e_1\};\end{aligned}$$

And the result of the transition by means of equation 3.5 is:

$$\begin{aligned}\gamma(dn_0, \rho_\delta(dn_0, n_1, m_traverse_nominal, \theta)) &= dn_1; \\ name &: dn_1; \\ id &: 1; \\ n_1 &= null; \\ N_1 &= \{n_2 : \langle \tau_{d_2} = [11, 20], v_{d_2} = Communicate(file_2) \rangle; n_3 : \\ &\langle \tau_{d_3} = [0, 10], v_{d_3} = Traverse(graph_3, p_{init_3}, p_{dest_3}) \rangle; n_4 : \\ &\langle \tau_{d_4} \text{ not grounded}, v_{d_4} = InitTravGraph(graph_4) \rangle; n_5 : \\ &\langle \tau_{d_5} \text{ not grounded}, v_{d_5} = FindRoute(p_{init_5}, p_{dest_5}) \rangle; n_6 : \\ &\langle \tau_{d_6} \text{ not grounded}, v_{d_6} = TraverseCycle(path_6) \rangle\}; \\ E_1 &= \{e'_1 : f_{temp}^{before}(n_3, n_2); e_2 : f_{temp}^{before}(n_4, n_5); e_3 : f_{temp}^{before}(n_5, n_6); e_4 : f_{temp}^{before}(n_3, n_4); e_5 : \\ &f_{temp}^{after}(n_3, n_6); e_6 : f_{param}^{equal}(graph_3, graph_4); e_7 : e_{a6} : f_{param}^{equal}(p_{init_3}, p_{init_5}); e_8 : \\ &f_{param}^{equal}(p_{dest_3}, p_{dest_5})\}; \\ dn_1^{sub} &= \{dn_a\};\end{aligned}$$

3.3 Properties of HTLN

In order to define the properties of HTLN and compare it with TRF the following assumptions are taken.

- HTLN: Given a problem network $dn_i = (n_i, N_i, E_i, dn_i^{sub})$, there is a complex decision goal $d_{vc} \in N_i$ decomposed in $dn_m^{dec} = (v_m, N_m, E_m, dn_m^{sub})$ with $m \in M_{vc}^{select}$.
- TRF: Given the same problem dn_i , a number of decisions N_m and relations E_m among them, equal to those in dn_m^{dec} , must be added to dn_i .

3.3.1 Soundness and Completeness

Assessing the soundness and completeness of HTLN is difficult from a theoretical point of view and highly depends on the specific planner implementation and the model at hand.

The demonstration of soundness is based on the definition of solution provided in Section 2.4.4 and extended in Section 3.3.2 to incorporate the concept of task refinement.

Given a planning system based on a set of resolvers, each oriented to solve a type of flaw, the planner is sound if every resolver is sound. Formally, it is expressed as:

$$\forall \rho \in P, \text{sol}(\rho, dn_i, D, P) = \top \Rightarrow \text{planner is sound} \quad (3.24)$$

The demonstration by contradiction is very straightforward. Suppose this assertion is false. Then, the plan π returned by the planner is not a solution. If it is not a solution, π must contain at least one flaw ϕ which has not been solved by the corresponding resolver ρ , therefore ρ is not sound, as it has returned a plan which is not a solution.

Assuming that for a candidate solution dn_i , any given unfolder is capable of checking whether the conditions of each decision are supported, that the scheduler can check whether all STP and CSP constraints are satisfied and that the timeline completer can verify that there is no gap in any of the timelines⁶. To evaluate the effect of HTN on TRF in terms of soundness, two inference rules are required:

R1 - If dn_i is a totally ordered, grounded and primitive decision network⁷ then $\text{sol}(\rho_\delta, dn_i, D, P) = \top$.

R2 - If dn_i is a non-primitive decision network and dn_k is a primitive decision network derived from dn_i by iteratively decomposing all the complex decisions such that R1 holds for dn_k , then dn_k is a solution.

Any algorithm that preserves R1 and R2 is sound and complete [64]. This definition will be expanded in Section 3.3.2 to incorporate the concept of partial solution.

Regarding completeness, in case the maximum depth and branching of the search space were finite, then an exhaustive algorithm that examines all the possibilities would be complete. As the domains for all the elements in TRF, and HTLN by extension, are finite, the maximum branching is limited. Moreover, the maximum number of decisions in the solution has an upper bound equal to the problem timespan as each decision takes at least one unit of time, therefore the maximum depth is also limited. In consequence, any exhaustive planner for TRF is complete. Even if the number of Values $v \in V$ or domain of the parameters were infinite, the planner can achieve completeness in case it uses the appropriate search algorithm for the unfolding step as shown in Section 2.4.1.

Focusing on HTLN completeness, the fact that there exists a sequence of ground operators that stripsolves the problem does not necessarily mean that an HTN planner must find it as a solution. In particular, unless there is a sequence of decompositions of the goal decisions that results in that ground operator sequence, the HTN planner will not find it. Thus, the completeness of an HTN planner has to be defined with respect to both the domain decisions as well as the set of methods [95]. If the model is complete, that is, all complex decisions can be decomposed into a finite set of primitive decisions, then R1 and R2 are preserved and the decomposer resolver is complete.

⁶ This assumptions depend on the specific resolvers and are not meant to be demonstrated here

⁷ Notice that the unfolder and scheduler are in charge of verifying that all constraints are satisfied.

3.3.2 Robustness

It is difficult to do a quantitative analysis of how the different novelties proposed improve the plan robustness. Two techniques have been envisaged to help increasing plan robustness in HTLN: HTN methods and sufficient planning.

HTN Methods

Multiple real domains present for any given problem several solutions from a formal point of view, while in practice most of them are not reasonable. Moreover, the users may have strong preferences about the types of solutions they are willing to accept.

The problem is that in these domains it is crucial to take into account the expert knowledge that determines which solutions are valid. HTN facilitate the coding of these knowledge in the form of methods that represent large, stable and high-quality plan fragments. HTN methods allow the users not only to specify what skeleton plan is used to achieve a goal, but also to specify a priori what causal dependencies must hold in the plan fragment. This shifts the focus from precondition establishment to plan merging [56, 95].

In consequence, assuming that the engineers in charge of the modelling phase have endowed the planner with high quality methods, the solutions provided by the planner would be expected to be more robust than others in which these expert knowledge would not be taken into account. As a particular example, during the FASTER field trials the team was able to configure some activities with different time ranges depending on the behaviours, for example path planning during the initial calculation or during a repair. That helped to prevent to a great extent failures due to time overrun (see further information on this regard in Chapter 6.).

Sufficient Planning

Traditional timeline planners search for fully justified plans which are:

- Complete: The plan always specifies how to proceed (the timelines have no gaps).
- Fully ordered: All the decisions are sequenced.
- Valid: All the constraints are satisfied.
- Grounded: All the variables have assigned specific values.

This definition might represent an unachievable condition in real-world problems. In order to create more robust plans in non-deterministic, dynamic and partially observable environments, more flexibility is required. HTLN is based on the concept of Sufficient Plan which represents an evolution of the idea presented in [71]. It is defined as follow:

Definition 46 (Sufficient Plan) *A plan is sufficient if all variables and relations are sufficiently grounded, all fully grounded relations are satisfied, all decisions are sufficiently decomposed and all the mandatory goals can be achieved for at least one specific instantiation of the sufficient plan.*

This definition represents an extension of the definition provided in [71], where a *sufficient plan* is fully defined up to a certain point called plan horizon, ignoring activities that fall outside it. In HTLN, any decision or constraint might be partially defined according to an initial definition. The different concepts involved in this definition are described in the following sections.

Sufficiently grounded

All decisions $d_v \in D^{all}$ and relations $f \in R$ that appear as goals in the problem must specify whether they should be grounded or not at planning time. A decision is grounded when its value and parameters are grounded; a relation is grounded when all the decisions of its scope (those affected by the relation) are grounded. A partially grounded relation has two important consequences: (1) The relation cannot be satisfied, (2) In case of temporal relations, the resulting dn is partially ordered.

Sufficiently decomposed

In order to determine whether a decision or network is sufficiently decomposed, two additional variables must be added to the definition of decision.

$mustDecompose(d_{vc})$ indicates whether or not d_{vc} must be decomposed (it is obviously ignored for v^p s). This property can be used to avoid planning on those activities for which some relevant information is missing.

$decLevel(d_{vc})$ describes the level of decomposition of the decision d_{vc} . It can be in one of three different levels:

- Not sufficiently decomposed (nsd): The decision requires further decompositions in order for the planner to generate a valid plan. By recursion, a complex decision d_{vc} for which $mustDecompose(d_{vc}) = true$ is not sufficiently decomposed if it has not yet been decomposed or if any of its subdecisions is not sufficiently decomposed.
- Sufficiently decomposed (sd): Indicates that the decision has been partially decomposed. By recursion, a complex decision d_{vc} is sufficiently decomposed if it has not been decomposed and $mustDecompose(d_{vc}) = false$ or in case $mustDecompose(d_{vc}) = true$, d_{vc} has been decomposed and none of its subdecisions is nsd but at least one is sufficiently decomposed. All the activities in a problem should be at least sd to define a valid plan.
- Fully decomposed (fd): A primitive decision d_{vp} is always fully decomposed. A complex decision d_{vc} is fully decomposed if, regardless of the value $mustDecompose(d_{vc})$, it has been decomposed and all its subdecisions are as well fd .

Formally:

$$\begin{aligned}
 decLevel(d_{vc}) = nsd \quad & mustDecompose(d_{vc}) = \top \wedge dec(d_{vc}) = \perp || \\
 & mustDecompose(d_{vc}) = \top \wedge dec(d_{vc}) = \top | \\
 & m \in M_{d_{vc}}^{select} \wedge \exists d'_{vc} \in dn_m^{dec}, decLevel(d'_{vc}) = nsd
 \end{aligned} \tag{3.25}$$

$$\begin{aligned}
 decLevel(d_{vc}) = sd \quad & mustDecompose(d_{vc}) = \perp || \\
 & mustDecompose(d_{vc}) = \top \wedge dec(d_{vc}) = \top | \\
 & m \in M_{d_{vc}}^{select} \wedge \nexists d'_{vc} \in dn_m^{dec}, decLevel(d'_{vc}) = nsd \wedge \\
 & \exists d'_{vc} \in dn_m^{dec}, decLevel(d'_{vc}) = sd
 \end{aligned} \tag{3.26}$$

$$\begin{aligned}
 decLevel(d_v) = fd \quad & d \in D^p || d \in D^c \wedge dec(d) = \top \wedge \forall d'_{vc} \in dn_m^{dec}, decLevel(d'_{vc}) = fd
 \end{aligned} \tag{3.27}$$

where $dec(d_{vc})$ is a boolean function that determines whether the decision d_{vc} has been already decomposed or not.

Example 9 Continuing with the example of the Primary Rover, imagine that the rover needs to perform two traverses: $Traverse_1(p_1, p_2)$ within the field of view and $Traverse_2(p_2, p_3)$ beyond the field of view. A human operator on-ground might be interested in marking the second traverse with the flag $mustDecompose = false$ because it is not possible at planning time to predefine a trajectory or make any assessment about the properties of the terrain in this situation. The initial problem, labelled as dn_0 is defined in PDL as follows:

```
f1 <fact> compMission.t11.Idle();
f2 <fact> compNav.t11.Idle();
...
g1 <goal> compMission.t11.Traverse(?_inTeam = FALSE);
g2 <goal, ND> compMission.t11.Traverse(?_inTeam = FALSE);
```

where the tag *ND* (Not to Decompose) is used by the parser to set the flag *mustDecompose*. In consequence, $decLevel(g2) = SUFFICIENTLY_DECOMPOSED$ and no decomposition will be attempted on this decision. Regarding $g1$, in this example where the Primary Rover traverses alone, the decomposer would choose the decomposition path shown in Figure 3.2. Imagine that at step i the decomposer ρ_δ creates the i – th problem network dn_i by decomposing the last element of the path, *BGenerateMapPR*. At this moment, all complex decisions in the network are decomposed in subdecisions whose depth level is 0. In consequence, $g1$ changes its decomposition level from *NOT_SUFFICIENTLY* in dn_{i-1} to *FULLY_DECOMPOSED* in dn_i . dn_i also changes its decomposition level value (which is equivalent to the lowest value among all its decisions). It corresponds to $g2$, which is *SUFFICIENTLY_DECOMPOSED*. At that moment, the plan is considered a solution from the point of view of the decomposer. If no other solving steps are required, then the planner returns dn_i as a solution and is ready to be executed. Later on, the problem will be completed by some deliberative agent on-ground or on-board by decomposing $g2$ in subdecisions.

This approach offers plenty of benefits. In scenarios where communications are not possible and the information required to complete some of the goals of the mission is still not available, sufficient planning allows to define the whole mission, marking these problematic goals as *NotToDecompose*, uplink the sufficient plan and let the replanner on-board complete it at due time. This situation is actually quite common in Mars missions as soon as the rover goes beyond the field of view. Even if it is possible to communicate with the robot, this technique can help in order to create look ahead plans. By defining some estimation of time and resource consumption for the sufficient goals, it is possible to generate a sufficient plan to get a better idea of how it looks like. The plan is even executable until it reaches the sufficient goals, that will have to be decomposed.

3.3.3 Performance

Analysing the performance of a HTLN solver is a complex task due to the interaction of different solvers with different properties. In the following subsections, it will be first theoretically demonstrated that the addition of a HTN step does not represent a penalty in terms of time or space complexity. Next, it is analysed how a HTLN-based solver might be benefited from an improved performance with respect to other TRF planners thanks to two fundamental HTLN characteristics: HTN methods and parallel planning.

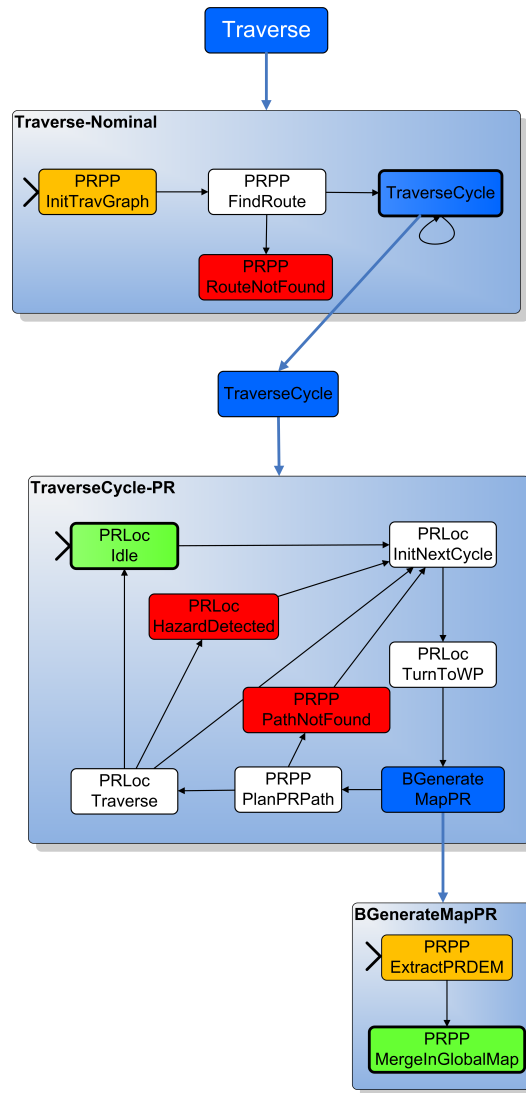


Figure 3.2.: Decomposition methods for *Traverse* complex goal in FASTER scenario.

Complexity Analysis

The complexity of a HTLN planner depends on the specific implementation of each of the resolvers. However, it is possible to analyse the asymptotic complexity of each resolver and specially the impact of HTN in the TRF formalism using Big O notation.

Finding an optimal plan in HTN is at least as difficult as the problem of determining whether or not a plan exists [64]. In HTLN, plan existence with partially ordered, not-regular decision networks is decidable because: (1) Plan length is finite: As the time horizon is finite and the minimum time duration for a decision must be greater than 0, only a finite number of complex decisions can be added to a given timeline; (2) Acyclic methods: Considering that the level of abstraction of any complex decision $level(d)$ in the domain D is limited by the upper bound k (see Section 3.2.5), given a set of acyclic methods, that is, for a decision $d' \in dn_d^{dec}$ it is guaranteed that $level(d') < level(d)$ where 0 is the level of all primitive decisions, then any decision can be expanded to a depth of at most k . In consequence, the number of possible decompositions for any given problem has always an upper limit and therefore plan existence is decidable from a HTN perspective [64, 62]. In addition, plan existence is also decidable for

models with cyclic methods as the number of decisions that can be allocated for any given plan is limited as indicated in point (1).

The process of unfolding and decomposition search are both based on the problem of searching in the tree (or graph for domains with loops). To characterize the time and space complexity, let us assume that b is the branching factor of any decision (the number of applicable actions or methods for the unfold and decomposer respectively), d the maximum depth of any path, ρ an unfold or decomposer resolver and greedy best-first the search strategy. The resolver ρ has a time and space complexity equal to $O(b^d)$ (see Section 2.4.1).

With respect to the scheduling problem [37], required to solve the associated CSP problem, consistency/filtering algorithms such as *PC* are more appropriate than solving algorithms because they can be used incrementally as new time points and parametric constraints are added. *PC-8* [46] has a time complexity of $O(n^3d^4)$ and space complexity of $O(n^2d)$ with no extra space required, being n the number of nodes and d the size of the domain. The discussion focuses on CSP complexity as it typically dominates the complexity of STPs which is $O(n^3)$ for the Floyd-Warshall algorithm. For APSI, the specific type of CSP problem is known as the Resource Constrained Project Scheduling Problem with Generalized Precedence Relations (or RCPSP/max), which is considered to be particularly difficult, more specifically NP-Hard [7], due to the presence of temporal separation constraints.

In consequence, the complexity of the scheduling problem theoretically dominates the complexity of the other steps or, in other words, the addition of a HTN step to the solving process does not asymptotically increase the overall complexity.

Another important aspect to consider in the complexity analysis is the depth at which each step happens. Regardless of the search strategy, the nodes at the solution layer of the search tree cannot correspond to those generated in the decomposition step. Subgoaling after decomposition is not mandatory, as it might happen that all the subdecisions added to the problem network are already fully supported. However, scheduling the new decisions added by the unfold or decomposer is mandatory. Considering the rather general assumption of a search tree with the same, or nearly the same, branching factor b at each level, it can be stated that there are more nodes in the last layer of the tree than among all previous layers together. In consequence, most of the work actually will be accomplished by the scheduler.

Performance Improvement - HTN Methods

The use of domain-specific knowledge can improve the performance of a planner from exponential to polynomial time [83, 141].

In experimental studies [115, 117, 4, 116], hand-tailored planners have solved problems orders of magnitude more complicated than those typically solved by planning systems based on domain-independent algorithms and knowledge. For example, in very simple problems with few constraints where the number of applicable actions is high, classical planning has huge problems due to the explosion of nodes in the search space. This kind of problems is very hard to solve with generic heuristics, but very simple in case domain-specific information is added in the form of HTN methods.

Adding a decomposition step in HTLN provides different advantages that might improve the overall solver performance.

Reduce the search space by decreasing the maximum depth

First, methods help to reduce the size of the search space as it decreases the maximum depth d by compacting several unfolding-scheduling steps in just one decomposition. Suppose for simplicity and

without loss of generality, that all resolving steps (unfolding, scheduling and decomposition) have a branching factor equal to b and that each new decision added during unfolding must be scheduled before others are added. Assume that for a given complex decision d_{vc} , there is a relevant method dn_{vc}^{dec} in which d_{vc} is decomposed that contains n decisions. In the worst scenario, the decomposer will need to expand all b methods to get to dn_{vc}^{dec} . On the other hand, a solver based on unfolding-scheduling steps would need b^d nodes, being $d = 2(n - 1)$ because we need two expansion levels (one for each resolver) to increase in one the number of decisions.

In addition, each method dn_{vc}^{dec} typically represents a carefully designed partial plan that grasp the knowledge of the experts and is therefore very optimized and stable. From the b^{2n-2} methods generated by the unfold-scheduler, all those that are not represented in the set of methods dn_{vc}^{dec} most probably will not represent good solutions, even if they are valid plans.

Reduce the search space by early pruning not valid solutions

Pruning partial plans is a more complicated topic. Using abstractions in planning does not guarantee an improvement in search efficiency. Tenenbergs [146] defined the so-called **Upward Solution Property**, which can be informally stated as: *“If there exists a concrete solution, then there also exists an abstract solution”*. Though this property is important, it does not establish a relationship between the abstract and concrete plan. Knoblock’s works [99] established a more restrictive property named **Ordered Monotonicity Property**: *“An abstraction space satisfies the ordered monotonicity property if any concrete solution can be derived from some abstract solution while leaving the actions in the abstract plan intact and relevant to the concrete plan”*. The ordered monotonicity property does not guarantee a good performance as it does not enforce that every abstract solution can be refined to a concrete one. This property is provided by the **Downward Refinement Property (DRP)** [5]: *“If a non-abstract, concrete level solution to the planning problem exists, then any abstract solution can be refined to a concrete solution without backtracking across abstraction levels.”*

If DRP is satisfied, at any given level of abstraction search for alternate abstract plans can be terminated once a single correct plan has been found because it is guaranteed that any abstract solution can be refined to a concrete level one without ever need to backtrack. In consequence, when a new operator is added to refine the plan, it is added solely to achieve the new (more detailed) preconditions, not to achieve previously satisfied conditions. Thus, if there is a concrete level solution, search in this manner will find it without having to consider alternate abstract plans [5]. DRP can rarely be guaranteed in actual planning domains. The way to achieve it in HTLN is to define the model in such a way that the domains of the elements on d_{vc} are an envelope of the domains in the subnetwork dn_m^{dec} such that: (1) The duration of d_{vc} must be an upper limit of the duration of all its possible decompositions; (2) The domain of any parameter in d_{vc} must always include the domain of any equivalent parameter in some decision $d \in dn_m^{dec}$. If the envelope property is respected, it is guaranteed that any decomposition into subgoals will never violate previously satisfied conditions and therefore DRP will be preserved.

Trying to satisfy DRP might have however consequences with respect to completeness due to the downward unlinearizability [95].

Definition 47 (Downward unlinearizability) *A constraint is monotonic if, once a plan is inconsistent with respect to this constraint, no amount of additional steps can make it consistent [94].*

Monotonicity of a constraint depends upon the types of refinements allowed by the planner. One effect of adding task decomposition to a temporal planner is that constraints that are normally monotonic might become non-monotonic, therefore they cannot be used to prune partial plans. As an example for HTLN,

if an abstract decision d gets a very wide temporal interval, it could make some abstract plans not valid, even though they could be refined into valid plans once d were decomposed.

Simplify the network by reducing the number of constraints

HTLN also has a direct theoretical impact in terms of performance with respect to TRF based on the number of constraints generated in the problem network. Consider the number of nodes $|N_m| = x$ and the number of edges $|E_m| = y$. Table 3.3 shows a comparison of the number of constraints that would be required in case the decisions and relations of dn_m^{dec} were added using HTLN and TRF.

dn_m^{dec}	Mode	Ordered	\neg Ordered
Ordered	HTLN	$y + 2$	$y + x$
	TRF	y	$y + 2x^*$
\neg Ordered	HTLN	$y + 2$	$y + x$
	TRF	m	m^*

Table 3.3.: Comparison of the number of relations in HTLN vs TRF.

In case dn_m^{dec} is ordered, HTLN requires two more relations than HTN to order the complex decision with respect to its subnetwork, as explained in Section 3.2.7. Addition of further decisions or relations in dn_i will have an equal impact regardless of the formalism used.

However, if dn_m^{dec} is not ordered, the number of temporal relations needed in HTLN is smaller. HTLN will require $y + x$ relations corresponding to the internal relations of dn_m^{dec} plus a *Contains* relation between the complex decision of the network and its subdecisions. If dn_i is ordered, TRF will require $y + 2x$ where $2x$ correspond to the temporal relations: (1) $f_{temp}^{before}(last, d_{m_i})$ between the *last* decision in dn_i before dn_m^{dec} and each decision in dn_m^{dec} ; (2) $f_{temp}^{after}(first, d_{m_i})$ between each decision in dn_m^{dec} and the first decision in dn_i after dn_m^{dec} . If dn_i is not ordered, the decisions of dn_m^{dec} will be completely mixed with the rest of decisions in dn_i and it will not be possible to tell that they pursued a common goal. This fact represents an important penalization in case further *Before* or *After* temporal relations had to be added to dn_i : In case a decision a of dn_i has to be ordered respect N_m , it would require in TRF x additional relations between a and each of the decisions in N_m while it would require only one relation in HTLN between a and dn_m^{dec} .

Besides the number of temporal relations required, there is an intrinsic advantage derived from using HTN in STP problems. Suppose that a complex decision d_{vc} decomposed in the network dn_d . Provided that the temporal relations of the decisions in dn_d are relative to the time points of d_{vc} (as explained in the Instantiation and Decomposition steps explained in Section 3.2.12), single modifications affecting the temporal relations in which d_{vc} is involved will immediately affect the temporal relations in the subnetwork dn_d without the need of any constraint propagation. This positive effect happens precisely because the temporal elements of the subnetwork are relative to those of the complex decision. For example, if d_{vc} is moved after another decision d_i , then dn_d will also be moved. In consequence, the performance of the scheduler should be improved as it requires less steps in order to obtain a solution.

Performance Improvement - Parallel Planning

By reasoning about the underlying graph of an HTLN problem, it is possible to identify separate subproblems, allowing the use of parallel planning. As explained in [54], a graph $G = (V, E)$ has a separation vertex v if there exist vertices a and b , $a \neq v$, $b \neq v$ such that all the paths connecting a and b pass

through v . A graph that has a separation vertex is called separable. Let $V' \subseteq V$, the induced subgraph $G' = (V', E')$ is called a non-separable component if G' is non-separable and if for every larger V'' , $V' \subseteq V'' \subseteq V$, the induced subgraph $G'' = (V'', E'')$ is separable. An efficient algorithm to generate valid temporal plans (not considering resource consumption) and computing the minimal network is to first find the non-separable components $C_1 \dots C_m$ ⁸ and then solve each one of them independently. If all components are valid, then the entire network is valid and the minimal networks of the individual components coincide with the overall minimal network. Taking advantage of HTLN structure, it is possible to isolate independent subproblems dn^{sub} and assign resolvers in parallel threads to each of them [170]. Further information about the specific implementation of parallelism for temporal networks in QuijoteExpress is provided in Section 4.3.5.

3.3.4 Expressiveness

HTLN provides several advantages in terms of expressiveness that can be summarized as:

- More structured way of defining domains in different abstraction layers.
- Easier plan inspection derived from the hierarchical organization of the solution.
- More expressive language to define complex/primitive decisions, meta-relations and loops.

Focusing on the domain definition, it could be argued that a hierarchical structure could better characterize the way in which humans think about problems [52]. Besides, a hierarchical model allows the user to organize the domain in different complexity levels up to the desired level of detail or granularity, which could be easily extended as required.

Hierarchies also help to better understand a plan from a human point of view, simplifying its verification and validation as the user can focus on a specific layer disregarding the details of lower ones.

As explained in the previous section, HTN allows the user to define domain-dependent knowledge. It allows domain-independent planners to be easily customized for different problems just replacing the set of methods.

From a theoretical point of view, HTN is more expressive than classical STRIPS-based planning and TRF as it is Turing-complete [63]: even undecidable problems can be expressed as HTN planning problems. While the set of solutions for a classical planning problem is a regular language, the set of solution for a total-order STN planning problem is a context-free language [64].

One consequence of using a context-free language is the power to represent loops, just by adding to the list of subdecisions dn_m^{dec} the reference decision v_m^{ref} in the method specification, as shown in the following example.

Example 10 *In FASTER, the `TraverseCycle` complex decision is in charge of moving the rover/s needs to iterate through a sequence of activities, whose main steps are: Perception, Map Generation, Path Generation and Traverse. However, the exact number of iterations required to reach the target is unknown beforehand. It can be expressed in HTLN as follows:*

```
DECOMPOSE compNavigation.t11 {
VALUE TraverseCycle(?inTeam, ?initWP, ?targetWP){
?initWP != ?targetWP;
cd1 compPrimaryLoc.t11.InitNextCycle();
cd2 compPrimaryPath.t11.LocateScout();
cd3 compScoutPath.t11.TransmitPose();
...
}
```

⁸ Notice that here the dissertation talks about graph components, not about APSI components.

```

cd13 compPrimaryPath.tl1.PlanPrimaryPath(?inTeam, ?nextWP);
...
cd17 compPrimaryLoc.tl1.TraversePrimary(?initWP, ?nextWP);
cd18 compNavigatatin.tl1.TraverseCycle(?inTeam, ?nextWP, ?targetWP);
}
}

```

The initial condition $?initWP \neq ?targetWP$; checks that the initial and target waypoints are different, in which case the loop is executed. After generating the map, the path is planned, obtaining the next intermediate waypoint $?nextWP$, used for the next traverse. Finally, the method calls itself again, replacing the previous $?initWP$ by $?nextWP$. This representation is more powerful than others based on quantified goals applicable in conventional TLP planning, which would require to know in advance the number of steps [95].

Moreover, HTLN allows to distinguish between complex/primitive goals. While the relation between complex goals and their associated methods is represented using the same structure as conventional APSI synchronizations, the meaning and consequences are completely different. A synchronization represents a relation between a decision (the reference) and a network. Synchronizations are part of the domain and involve decisions that are not required to be in different levels of complexity. During planning, enforcing a synchronization implies the need to satisfy the set of constraints (temporal or parametric) between the reference decision and the target decisions.

On the other hand, the methods do not belong to the domain, but to the knowledge database. In a method, one decision must be in a higher level of complexity than the network in which it is decomposed. Finally, applying a method in HTN implies the replacement of the complex decision by the set of subdecisions in the network.

Besides the representation of loops and complex/primitive decisions, HTLN allows to express meta-relations between a decision d_v and a sub-network dn_i thanks to the possibility of binding together a decision network under a higher level complex decision. Assuming that E_m does not totally order the nodes in N_m , it is possible in HTLN to specify any of the relations *Meets/Overlaps/During/Starts/Finishes/Equal* between a decision $a \in dn_i$ and the decomposed network dn_m^{dec} (interpreted as a unique complex decision), but it is not possible in TRF as there is no complex decision encompassing all the subdecisions to which the relation could make reference.

Example 11 Given a problem in which the Primary Rover must perform a *Traverse* and then *Communicate to Earth*, suppose that the goal *Traverse* has been already decomposed in the network $dn_{Traverse}^{dec}$. The following constraint expresses the temporal ordering between the subnetwork obtained from the decomposition of *Traverse* (which is a *DecisionNetwork*) with *Communicate*, which is a simple decision.

```
f_{temp}^{before}(Traverse, Communicate);
```

3.4 Conclusions

This chapter has presented HTLN, a formalism that combines temporal and task networks in an effective way. Taking advantage of the formal definition, it was possible to prove that HTLN offers several advantages with respect to conventional TLP and HTN in the fields of performance, plan robustness and expressiveness.

The increase of performance comes from a better time and space complexity derived from the use of partial plans (the methods) plus the capability of performing parallel planning.

In the field of expressiveness, the main advantage of HTLN comes from a novel approach to HTN planning in which the hierarchical order between the complex decision d_{vc} and its subdecisions in dn_{vc}^{dec} is maintained in the network after decomposing d_{vc} thanks to the dual representation of complex values as decisions and networks. That allows to express relations between any other decision in the problem network and the whole subnetwork dn_{vc}^{dec} . The use of methods also allows an elegant way to express recursion in a method m by including in the target network dn_m^{dec} a call to the reference decision v_m^{ref} .

Regarding robustness, HTLN contributes in two different ways to improve it, sufficient planning and HTN methods. The first provides a tool to construct more robust plans in scenarios with high uncertainty, as it allows to postpone the decision making process to the time in which the information is available. HTN methods on the other hand are good quality sub-plans thoughtfully evaluated.

All this properties are theoretical. Chapter 4 presents a planner based on HTLN that will show their impact in real applications.

4 QuijoteExpress Planner: A Novel Planning System

Several factors, identified in Section 2.7, have prevented so far the use of more autonomy on robotic systems in the open-world. The main motivation of this thesis is to provide a novel temporal planner prototype for on-ground/on-board robotic missions in partially observable, non-deterministic and dynamic scenarios (see Section 2.2.2). More specifically, the aim is to improve a number of features identified in Section 2.2.2:

- **Performance:** It is critical in situations where a fast reaction determines the success of failure like Earth observation satellites that need to rapidly react to capture images of a specific region [137, 45] or for Mars rovers to capture serendipitous events such as dust devils [28]. In the rescue scenario, fast reaction is mandatory, as the survival rate rapidly decrease with time [143].
- **Robustness:** The planner should be able to produce robust plans even for highly complex scenarios were the assumptions presented in Section 2.2.2 hold.
- **User/planner interaction:** It can be improved in two ways: Increasing the language expressiveness to create more realistic models and generating more understandable plans.

The approach presented in this chapter is based on the combination of classical and applied planning techniques. The result is a novel planner called QuijoteExpress [174, 167, 175]. QuijoteExpress (abbreviated QE) is a timeline planner based on the HTLN formalism. It profoundly deviates from any other timeline planner developed so far, as it uses heuristic, forward-chaining search in the state-space as most classical planners.

The chapter is organized as follows. It starts with the description of the general architecture in which APSI* and QE together with APSI are put in context. Next sections present further details about the design and implementation of APSI* and QE. The planner performance is also evaluated in this chapter as a standalone component, comparing the impact of the different features with synthetic examples while Chapter 6 evaluates the whole system (including the Executive) in a real-world scenario. The chapter ends with the conclusions.

4.1 Architecture

The formalism introduced in Chapter 3 presents radical differences with respect to APSI. In order to produce a planner able to exploit both of them, an architecture organized in three layers has been designed.

APSI3 is in the lowest layer. Some modifications were required in the core of the framework, such as the representation of decomposition levels or primitive/complex decisions. These modifications have been now integrated in the official APSI distribution.

To implement HTLN as a general solution, able to be reused for other planners (including APSI ones), an additional abstraction layer, called APSI* had to be defined on top of APSI. APSI* follows the same internal organization dictated by APSI with the intention of maintaining a cohesive design, based on the organization of the software in a TRF* (Framework and Kernel) and Solver packages.

Finally, the planner sits on top of APSI and APSI*. It uses the STP and CSP capabilities of APSI and the solving approach (search space, resolvers, etc) of APSI*. The building blocks are presented in Figure 4.1.

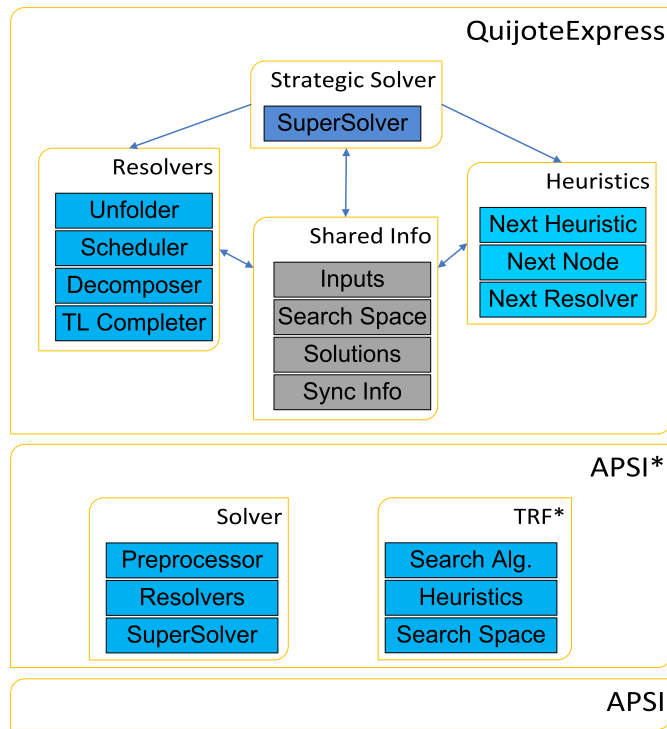


Figure 4.1.: Building blocks of QuijoteExpress and APSI*.

4.2 APSI*

APSI* is a new solving layer that extends APSI3 TRF to incorporate HTLN. As indicated in Section 3.2.12, the key concept for APSI* (derived from HTLN) that differentiates it from other HTN planners is the way in which complex goals are represented and managed. Most HTN planners like SIPE-2 [160] or O-Plan [145] replace the complex task in the problem by the set of subtasks of the method selected. In consequence, some information is lost as the problem does not retain the hierarchical structure of the domain. This information can be in fact useful during planning as depicted in Section 3.3.4.

APSI* is meant to be used as an abstraction layer, backwards compatible with APSI3, that isolates the user from low level details to facilitate the development of new HTLN solvers in any kind of search space (plan-space, state-space, etc.), so that any planner developed for APSI could use it. It focuses on providing functions and data structures to improve the performance, plan robustness and expressiveness of the planners.

From an architectural point of view, APSI* is placed on top of APSI3. It is divided into two layers, TRF* and Solver (Figure 4.1). As in APSI TRF, TRF* is organized in a low level of detail package named Kernel and a high level named Framework, which is actually where all functionalities for the user are placed. TRF* extends the original TRF in two directions:

- Algorithms: Define the general structure for solvers, heuristics and its related parameters.
- Search Space: Contain data structures defining what is a search node, open-list, closed-list and search-space.

The solver package does not provide any concrete class, but rather a set of abstract classes that define what is required to implement HTLN solvers. It is divided in three blocks: Preprocessor, Resolver and Supersolver which will be analysed in depth in Section 4.3.2.

4.2.1 Expanding the APSI Core

Adding HTLN requires some modifications in key components of APSI3. In the lowest level, the APSI parser in charge of reading the domain description files (DDL) was modified to be able to handle decompositions by means of the new keyword *DECOMPOSE*, which shares the same syntax as *SYNCHRONIZATION* but is computed by the decomposer resolver ρ_δ in a different way. It has been also added the keyword *DEFAULT* to indicate which is the default state of a *StateVariable* component. This property can be used as a landmark by the heuristics to estimate the distance from the present state to the goal. With respect to the problem description language (PDL), a new keyword *ND* has been added to indicate that its associated goal does not need to be decomposed. This keyword can be used by the planner on-ground to generate partial plans. All these concepts are illustrated in the following example.

Example 12 *In the FASTER scenario, Idle must be represented as the default state for the Mission component. Moreover, the method that decomposes the complex decision Navigate needs to be modelled. The corresponding DDL is showed below.*

```
COMP_TYPE STATE_VARIABLE MissionType
VALUES
{
DEFAULT Idle() [1, +INF];
Navigate(bool ?inTeam) [2000, +INF];
...
}

DECOMPOSE compMission.t11 {
VALUE Navigate(?inTeam){
cd1 compPrimaryPath.t11.InitTravGraph();
cd2 compPrimaryPath.t11.FindRoute();
cd3 compNav.t11.TraverseCycle(?inTeam3);

cd1 BEFORE [1,+INF] cd2;
cd2 BEFORE [1,+INF] cd3;

?inTeam = ?inTeam3;
}
}
```

Regarding the problem definition, the Primary Rover needs to perform two traverses (see model in Section 6.3.3), with the second being beyond the field of view. It might be desirable to generate a plan containing both traverses in order to get a better overview of the whole mission, but leaving the details of the second one undecided until the end of the first traverse, allowing the robot to gather more information before it is further detailed.

```
f1 <fact> compMission.t11.Idle();
f2 <fact> compNav.t11.Idle();
...
g1 <goal> compMission.t11.Navigate(?_inTeam = TRUE);
g2 <goal, ND> compMission.t11.Navigate(?_inTeam = TRUE);
```

With respect to TRF, the class *Value* contains a new flag to indicate whether it is primitive or not. This flag might change for the same value depending on the set of methods provided (known as Knowledge DataBase or *KDB*). For example, in the rover example presented in Chapter 3, the user might not define any method to decompose *Traverse*, in which case it would be primitive, leaving the responsibility of its interpretation to the Executive. In case a method such as *m_traverse_nominal* is defined,

Traverse becomes complex. To avoid inconsistencies, the preprocessor is in charge of computing the primitive property for each value of the domain in advance. Another flag *mustDecompose* is required to indicate to the planner whether a complex decision d_{vc} (primitive decisions cannot be decomposed) needs to be decomposed or not (see Section 3.3.2). Because it is not possible to clone a decision in APSI, in case two different search nodes share the same decision d and one of them modifies it, the changes will also be reflected in the other node. The implications for HTLN are important, as different nodes might have different levels of decomposition for the same decision. To solve this problem, this parameter becomes intrinsic to the *DecisionNetwork* (DN). Each network contains a table with the level of decomposition for all its decisions plus its own decomposition level which is computed every time a new decision or subnetwork is added according to equation 3.27.

The application layer has also been modified to process the *KDB* and to use the preprocessor.

4.2.2 Search Space

The *SearchNode* is the data structure that contains all the relevant information related to a given state of the world. It is the main unit of information of the search space.

A search node contains a state, which type depends on the type of node. Besides, some bookkeeping information related to the search process and the state itself is stored:

- Id: Unique identifier of the node.
- Solver: The solver that generated the node.
- Parent: For a given node n expanded in a number of children $\{n_i\}$, n is the parent of n_i .
- Status: A node can be *New* when it is created as a result of an expansion, *Open* while it is being analysed by a solver for expansion or *Closed* once it has been expanded. Closed nodes do not need to be re-open in case the search is complete and generates all possible children, as it happens in QE. For other planners such as AP² [36], if no solution is found, a backtracking process is required to go back to a previously closed node, and then re-open and expand it again.
- Solution type: In case it has not been checked whether or not the node is a solution, its type is *NOT_EVALUATED*. This is the case of some special nodes such as the first node generated from the initial state defined in the problem or the node generated after the combination of several subproblems. Every time a node n is open, it is evaluated whether the node is a *SOLUTION*. In case it is not, n is expanded by a solver (Supersolver or Resolver). If n has no children, then it is a *DEAD_END*. A solution node n_i returned by a resolver can be: A *RESOLVER_STEP* if it still requires other resolvers; A *PARTIAL_SOLUTION* if n_i does not require any other resolver but is just a solution for one of the subproblems in which a higher order problem was divided; A *SOLUTION* otherwise. The cycle relating status and solution type for a node is presented in Figure 4.2.
- Alive children: In case the problem contained in the node n has been divided in several subproblems, this variable indicates how many subproblems are pending to be resolved.

APSI* provides two type of nodes: *SimpleSearchNode* and *SuperSearchNode*. Before describing how they are used, the concept of frontier must be introduced.

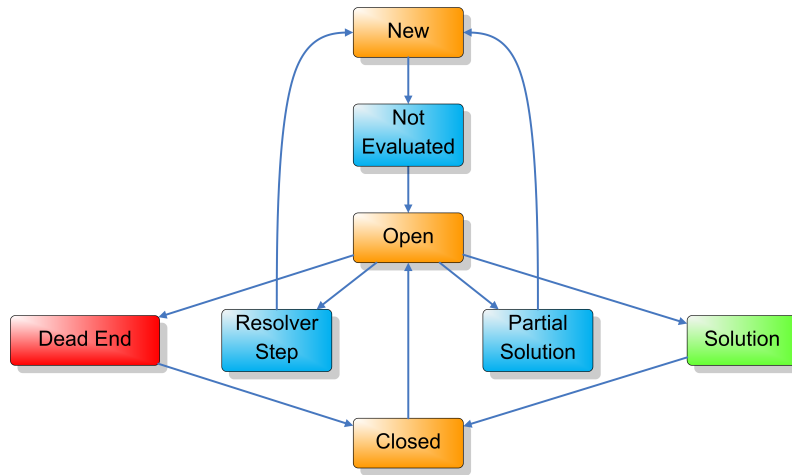


Figure 4.2.: Relation between states and solution types for search nodes in APSI*: Orange: Different status of the node; Blue: *RESOLVER_STEP* solution type; Red: *DEAD_END*; Green: *PARTIAL_SOLUTION* or *SOLUTION*

Definition 48 (Frontier) *Each timeline has associated a **frontier** containing two decisions: (1) Last decision of the current state with respect to its temporal ordering; (2) First decision of the goal state with respect to its temporal ordering. Each timeline might have multiple frontiers, dividing a sequence of temporally ordered goals g_1, \dots, g_n in $n - 1$ subproblems where the current and goal state of subproblem $_i$ are g_i, g_{i+1} (see Section 4.3.3).*

The SimpleSearchNode is used by resolvers. Its state-type is a *DN*. It also contains a delta *DN* that indicates the additional decisions/relations added by the solver plus a frontier ¹

The SuperSearchNode is used by the Supersolver. Its state-type is a *SearchProblem* that contains an initial and goal SimpleSearchNodes, which represents a major difference respect conventional APSI planners in which the problem is defined by one single node that contains the facts and goals. Besides, it contains information required in case the SearchProblem has been divided in several subproblems.

Each solver (Supersolver or Resolver) has associated a search space used to support the search process as described in Section 2.3.1. A search space typically contains two lists of nodes: the open-list and the closed-list. The solver peeks a node n from the open-list and expands it, generating a number of children derived from n by means of some transformation performed by the solver. The children are stored in the open-list while n is moved to the closed-list. To improve the efficiency during search, most planners order the open-list according to one or several priority criteria heuristically evaluated. The problem with priority-queue open-lists is that the queue needs to be sorted every time an element is queued/enqueued, a process that might be computationally expensive when the number of nodes is large. To avoid this problem, the open-list is implemented as a two-levels bucket-based priority queue (Figure 4.3). First of all, it is required to obtain the range of f and g values, named here as $\{f\}$ and $\{g\}$ respectively. The first level of the queue is divided in a number of buckets corresponding to a range of values in $\{f\}$, each one subdivided in another queue with a list of buckets corresponding to a range of values in $\{g\}$. The second level is key to break ties in case two nodes receive the same f -value. APSI* provides two types of open-lists: single-queue and multi-queue (see Section 4.3.5).

In case there are loopy paths that can lead the search to previously explored nodes, the topology of the search space is a graph, in which case the closed-list is added to store all the nodes already visited.

¹ The frontiers are computed by the class *FrontierManager* provided by APSI*.

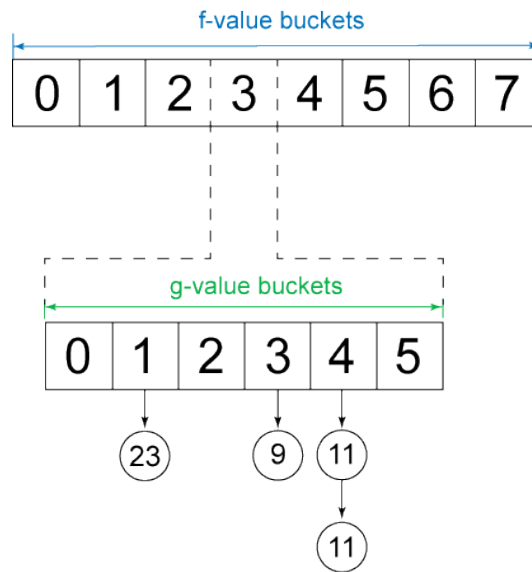


Figure 4.3.: Two-levels bucket-based priority queue (Courtesy of IBM).

Every new child generated that matches any node in the closed-list is discarded instead of being added to the open-list. The closed-list does not need to be ordered, therefore the addition/deletion of nodes is faster. In case there are no loops, the topology is a tree and the closed-list is not required.

The edges of the graph or tree connect a given node n with its successors n_i (if any), and contain information about the modifications done to the parent in order to produce the child.

Example 13 *Figure 4.4 illustrates the structure of the open-list. Suppose a problem n decomposed in two subproblems $n1$ and $n2$. The subproblem $n1$ is later on subdivided in $n11$ and $n12$. Once a solution is found for $n11$ and $n12$, they are recombined producing several solutions $n1s$ for the node $n1$. The same process is accomplished with $n2$. Once all the subtasks of n have produced subsolutions, a global solution ns is extracted.*

APSI* provides two different search spaces: the *SuperSearchSpace* used to store *SuperSearchNodes* of the *Supersolver* and the *SimpleSearchSpace* which stores *SimpleSearchNodes* of the *resolvers*. Both contain an open and closed list, so they can be used for graph and tree search.

4.2.3 Solvers

Most classical planners have two types of solving algorithms: search algorithms and heuristics. In the case of TLP, the concept of search algorithm needs to be generalized.

Definition 49 (Solver) *A solver in APSI* is an algorithm that, given as input an initial and goal state, tries to reach the goal from the initial state by means of some internal mechanism.*

A solver is endowed with an internal search space to store the intermediate nodes generated during the process. There are two types of solvers in APSI*: *supersolver* and *resolver*. Unlike the search space package, APSI* does not provide any specific solving algorithm but rather a list of abstract classes illustrated in Figure 4.5 that needs to be extended to create a new planner.

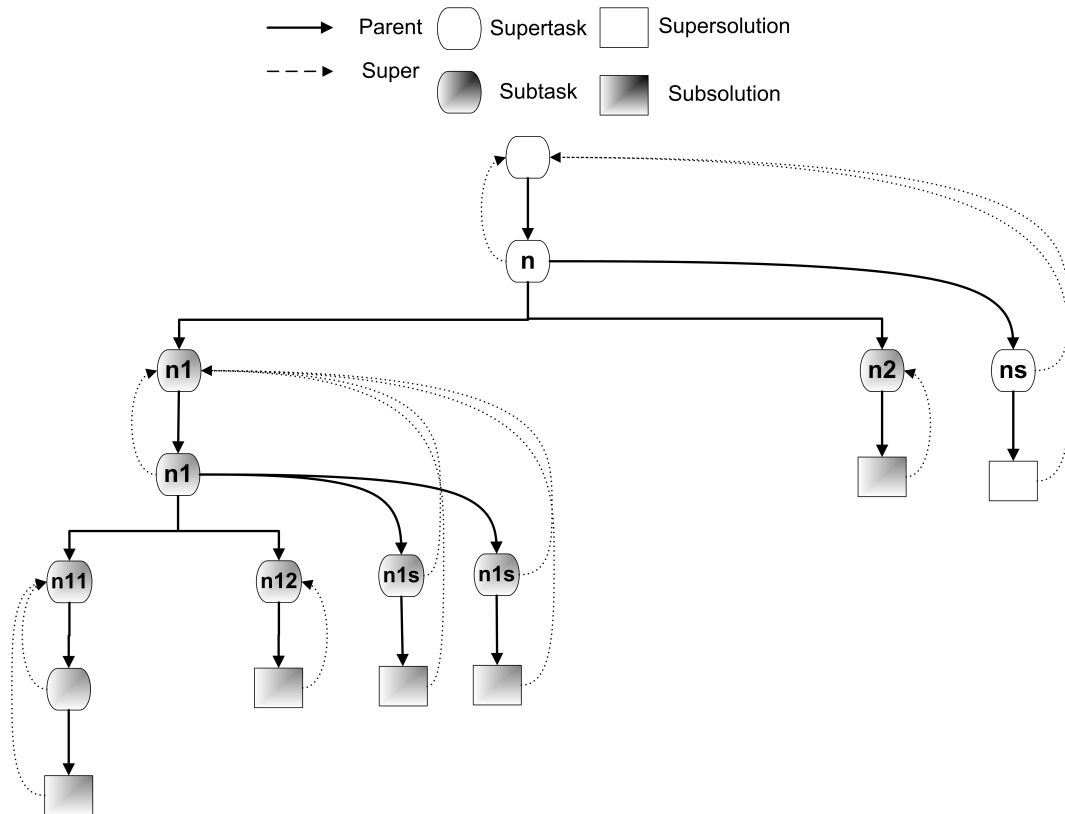


Figure 4.4.: Open list structure.

Classical planning consists of one single step, the expansion of a given node by means of all possible applicable actions to generate a number of children. In TLP however, a number of steps must be interleaved, traditionally including unfolding, scheduling and timeline completion, plus decompositions in the case of QE. As a consequence, a strategic planner, known as the **Supersolver**, is required to choose the next node to be analysed and the resolver to be used. APSI* approach is based on a Producer-Consumer paradigm where the **Supersolver** produces new jobs, that is, the nodes in the open-list to be consumed by different resolvers, each running in a separate thread. APSI* provides the general mechanisms required by the Supersolver to initialize the planner, launch resolvers in new threads and manage the search space, while QE (see Section 4.3.2) implements an instantiation of the Supersolver class.

The **resolver** is a type of solver used by the Supersolver to progress in the search. There are four types of resolvers in APSI*: unfolder, scheduler and decomposer. The results are sent to the Supersolver, which will be in charge of the analysis.

In terms of **heuristics**, APSI* just provides some abstract classes to be extended in the concrete implementation.

Heuristics and solvers can be launched in parallel as independent threads. However, some information needs to be shared among the resolvers and Supersolver to guarantee an appropriate synchronization, compiled in a number of thread-safe parameters in the class *PlannerSharedInfo*. First of all, *PlannerSharedInfo* takes care of coordinating the access to the *DomainManager* of APSI, in charge among other things of propagating the temporal constraints of the underlying STP network. It also contains a pool of resolvers and heuristics that can be launched in parallel at any time by the Supersolver². Other variables are used to inform the Supersolver every time a resolver finishes.

² the maximum number of solvers to be run in parallel is configured via a parameter of the planner

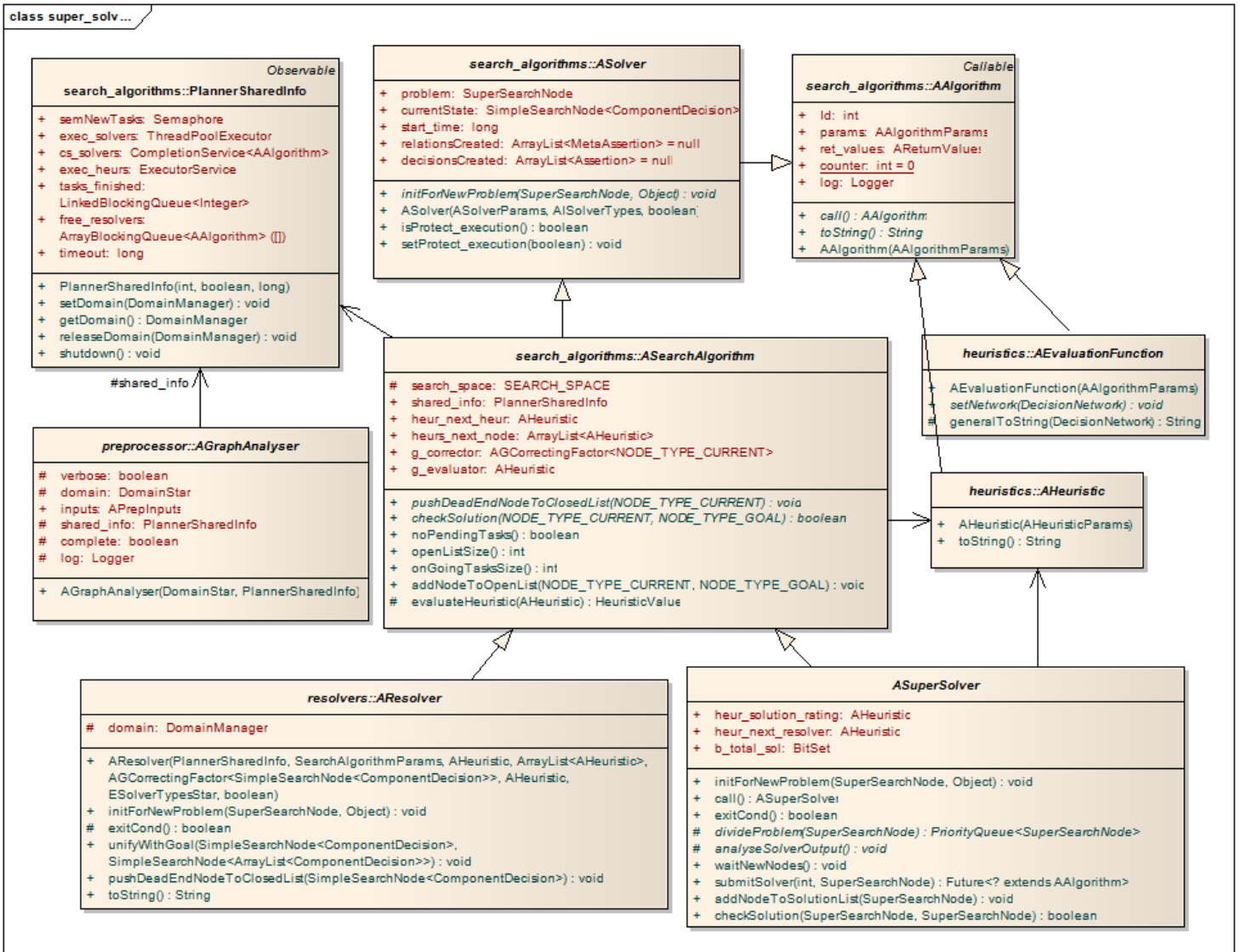


Figure 4.5.: APSI* algorithms class diagram and shared data.

4.3 QuijoteExpress

QuijoteExpress [174, 167, 175] is a planner based on APSI*. The planner is sound and complete for some configurations of the specific resolvers presented in Section 4.3.3. Nonetheless, it might not be possible (nor desirable) to guarantee these properties under some circumstances, e.g. when anytime replanning is required to provide a solution as soon as possible, in which case completeness might be compromised.

The planner benefits from all features provided by APSI, including the model-based approach, the use of DDL and PDL languages, which extensions for HTLN were presented in Section 4.2.1, STP reasoning by means of the temporal network and CSP using the Choco resolver. It also uses the scheduler provided by AP², even though some modifications were required (see Sections 4.3.3 and 4.3.3).

Other features of APSI has been disregarded in favour of their counterparts in APSI*, more specifically the search space, resolvers, heuristics and Supersolver. The novelties presented with respect to other TLP planners, some derived from the use of HTLN and analysed from a theoretical point of view in Chapter 3, are briefly listed below and further studied in subsequent sections:

- HTLN: The planner is based on the formal definition of HTLN planning with the aim of providing a more expressive language to describe domains and better understandable solutions in the form of hierarchical plans. Opposite to most HTN planners, QEv2 is based on the construction of hierarchical structures rather than goals replacement, adding new levels of detail as complex tasks are refined into subtasks [170] (see Section 3.2.12).
- Sufficient Planning: The planner can generate partially defined plans in order to produce more robust and versatile solutions as it postpones the need to take some decisions to the moment when the information is available.
- Parallel Planning: QEv2 can take advantage of modern microprocessors endowed with multi-threading, multi-core capabilities by running in parallel several resolvers and heuristics to improve the performance.
- Forward-Chaining in the state-space (see Section 4.3.3): QEv2 is the first heuristic forward-chaining TLP planner to the best of our knowledge. The aim is twofold: (1) To improve the performance of the planner and (2) To benefit from the huge amount of research accomplished in this field.
- Heuristic Planning: Derived from the previous point, QEv2 can exploit classical heuristics such as Landmarks or PDBs for satisficing and optimal planning in the state-space in opposition to Fewest Alternative First (FAF) heuristics typically used in TLP planning. The benefits are identical to those stated above.

Even though some of these aspects have been previously explored for classical planning, they are certainly innovative in the field of timeline planning. The combination of Sufficient Planning and HTLN might represent the first steps towards more advanced levels of autonomy (see Tables 1.1 and 4.1).

Level	Description	Function
E5	Execution of on-board reconfigurable plans	Allows the mission planner to autonomously modify the plan, dropping or adding new goals in support of special needs such as opportunistic science.

Table 4.1.: New Autonomy Level.

The E5 level implies an important change of paradigm in operations: while E4 is based on replanning a set of given high level goals, E5 gives the planner the capability of generating its own goals and discarding given ones. This technique, named **auto-goaling**, provides the flexibility required for missions in which the goals are unknown beforehand.

Regarding HTN planners, even though they present a rich variety of relations and constraints, TLPs are clearly more expressive in terms of time representation. The approach taken by HTLN, specially the dual representation of complex goals as networks and decisions (Section 3.2.4) and the way in which goals are decomposed (Section 4.2) is completely different to other HTN planners such as SIPE-2 [160] or O-Plan [145].

With respect to temporal planning, there are several planners in the literature (see Section 2.4.4) which scope is similar to QEv2. From all these planners, only IxTET and ASPEN are endowed with some kind of HTN capabilities. In the case of IxTET, the task decomposition is just a user programming facility to build models of complex tasks on the basis of simpler ones [101]. ASPEN is the closer approach to QEv2, but they present some key differences listed in Table 4.2, being the most important related to the

planning and solving strategies. Moreover, while QE keeps the hierarchical relation between decisions once they are added to the problem network, ASPEN behaves as other conventional HTN planners, replacing the complex activity by subactivities. QE supports parallel planning based on the analysis of frontiers while ASPEN does not have parallelism. Finally, both planners can specify that an action does not need to be decomposed, but QE allows more flexibility, distinguishing between full, sufficient and not sufficient decompositions.

Search	QuijoteExpress	ASPEN
Planning Approach	State-Space	Plan-Space
Solving Strategy	Forward-chaining	Iterative repair
Hierarchical representation	Inside the Decision-Network	Outside the network in the Activity Database (ADB)
Parallel planning	✓	✗
Sufficient planning	✓	Partially

Table 4.2.: Comparison of QEv2 and ASPEN.

4.3.1 Configuration & Preprocessor

A number of new parameters have been added to configure the solvers and heuristics.

- *maxThreadsSolvers*: Indicates the maximum number of resolvers of the same type that the Supersolver can launch in parallel.
- *maxNumSolutions*: In case a solver finds a number of solutions equal to *maxNumSolutions*, it finishes and returns.
- *timeOut*: Maximum time allocated for the Supersolver to return solutions.
- *complete*: Configures the solvers to explore all the nodes until *maxNumSolutions* is reached, the *timeOut* is reached or the search space is exhausted.
- *decompLevel*: Indicates the minimum decomposition level (*SUFFICIENT* or *FULL*) reached by the solutions.
- *decomposer, unfolder, scheduler*: Specifies the algorithm used for each type of resolver.
- *singleEvalPerNode*: Even if multiple heuristics are defined, only the heuristic selected by *metaHeur* will be evaluated. It should be used to save time required to evaluate heuristics.
- *deferredEval*: Assign to a child node the heuristic value of the parent. Should be used if heuristic computation is more expensive than memory space.
- *gCorrectionFactor*: If *deferredEval* is used, then the child's *h*-value is corrected by subtracting to the parent's *h*-value the (mean/specific) *g*-value of each successor node.
- *nextResolver*: Specifies the strategy to be used by the Supersolver to select the next resolver.

- *metaHeur*: Specifies the heuristic used to select the next-node-heuristic, in case several are defined.
- *heurNextNodeUnfolder*, *heurNextNodeDecomposer*: Indicates one or several heuristics together with their weight to select the next node for the unfold and decomposer.

The preprocessor is derived from the work done in RoBen [173, 168], a benchmarking tool used to analyse the complexity of the problems. It is launched before the planning phase to extract knowledge about the domain and problem in a similar way as [86] does. The purpose of this information is to be used later on by heuristics and resolvers to speed up search. Specially relevant is the computation of the temporal transition matrices and the value's lowest hierarchical level.

Temporal Duration Transition Matrix

For each StateVariable in the domain, the transition temporal durations from each state to the rest of the same component are calculated. These values are intended to be used by heuristics to estimate the best path from the initial state towards the goal in a fast way. Therefore, the way to calculate the transition duration must be optimistic in order to guarantee admissibility. This is achieved by means of the following assumptions:

- Given a value v which duration is defined as an interval $[min, max)$, the duration assumed is always the lower bound.
- Given a transition from the value v_i to v_g which duration is defined as an interval $[min, max)$, the duration assumed is always the lower bound.
- The instant duration from a value to itself is its minimum duration.
- If a decision is complex, its minimum duration is equal to the minimum timespan among all its target networks.

The result is a matrix having as columns and rows the list of all the values defined in the domain where a given cell $[v_i][v_g]$ indicates the duration from v_i to v_g .

Decision Lowest Hierarchical Level

As mentioned in Sections 3.2.5 and 3.3.3, all decisions in the domain have associated a hierarchical level related to the structure defined by means of the methods M in the domain. Algorithm 1 shows how the hierarchical level is computed.

The algorithm first makes a copy of all primitive values $V^p \in V^{all}$ of the domain in V_{next} . The initial values (primitives) get assigned a level 0 and then the algorithm enters a loop in which the next level of values is obtained as follow: Given the current list of values V_{next} , all the methods in the domain which target network contains at least one value $v \in V_{next}$ is extracted in M_{next} . The next hierarchical level corresponds to the list of reference values v_m^{ref} of M_{next} . The process is repeated until the list of methods is empty. Unlike for the generation of the causal-graph in planners such as FastDownward, this algorithm does not need to make any relaxation in the domain to handle loops, as they are avoided in a natural way following and ascending search using the network-reference organization of the methods, which is monotonically decreasing in number of values. The hierarchical order is fundamental for the decomposer to define the order in which complex decisions must be decomposed (see Section 4.3.3).

Algorithm 1: `preprocessor.computeHighestHierarchicalLevel()`

begin

```
next_level ← 0
V_next ← Vp
level(v_next) ← next_level, ∀v_next ∈ V_next
M_next ← M_{v_next}^{select}, ∀v_next ∈ V_next
while (M_next ≠ ∅) do
  next_level + 1
  V_next ← v_m^{ref}, ∀m ∈ M_next
  hlevel(v_next) ← next_level, ∀v_next ∈ V_next
  M_next ← M_{v_next}^{select}, ∀v_next ∈ V_next
```

4.3.2 Supersolver

The strategic solver, hereafter referred to as Supersolver, inherits the basic functionalities provided in APSI*. It takes care of three fundamental activities: Division/recombination of a problem into subproblems, the solving loop and analyse the outputs of the resolvers.

A Supersolver receives as inputs a SuperSearchNode with an initial and goal state, shared parameters for synchronization purposes, configuration parameters, a list of one or more heuristics to evaluate the h -value of supernodes, a heuristic to rate the solutions, an algorithm to choose the next resolver and a g -evaluation function to determine the cost of the node. It returns a list of solutions sorted according to the solutions-rating heuristic.

Algorithm 2 presents the solving loop:

Algorithm 2: `Supersolver.call()`

begin

```
openList ← divideProblemInSubProblems(problem)
while (¬exitCond()) do
  if (pendingResults(sharedInfo)) then
    analyseResults()
  if (openList.size() > 0) then
    node ← selectNextNode(openList)
    ρ ← selectNextResolver(node)
    submitSolver(ρ, node, domain)
  else if (onGoingJobs(sharedInfo)) then
    waitResults()
```

Initially the Supersolver tries to split the problem into subproblems (see Section 4.3.5), adding the result to the open-list of the search space and evaluates the exit conditions of the loop which logic is presented in Equation 4.1.

$$\begin{aligned} requestedStop \vee pendingJobs = 0 \vee numSolutions \geq maxSolutions \vee \\ timeConsumed \geq timeout \Rightarrow exit = \top \end{aligned} \quad (4.1)$$

In case the exit conditions are not satisfied, then it checks whether any resolver has finished, in which case it analyses the output. Otherwise, in case the open-list is not empty, it heuristically selects the next

node and a resolver for it and launches the resolver in a new thread. If the open-list is empty, then it waits until any of the resolvers returns a solution and the loop starts again. There are four fundamental concepts inside this algorithm: the analyses of the resolver's output, the division of a problem into subproblems, the selection of the next resolver and the selection of the next node. The first three are studied next, while the selection of the next node will be studied in Section 4.3.4.

Selection of a resolver

The order in which resolvers are selected can play a relevant role in the overall performance of the planner. The strategy used by QEv2 to choose the next resolver is determined by means of an external algorithm to be specified in the configuration file. The ordering is not totally flexible and depends on the characteristics of the different resolvers. In the case of QEv2, for example, a limiting factor is the need to run the scheduler when the decisions are not totally ordered prior to the execution of the unfolder, as it needs the DecisionNetwork to be totally ordered to extract the frontiers.

Two strategies illustrated in Figure 4.6 and 4.7 have been conceived. Both of them start by verifying whether the node is a solution, in which case the timeline completer is called. Otherwise, the unfolder is called. The main difference lies on the next step. Assuming that the problem network is not sufficiently decomposed, *UDS* would then call the decomposer, while *USDS* interleaves a scheduling step between the unfolder and the decomposer. Either way, the scheduler is always run before the next unfolder execution and the decomposer is only executed if the network is not sufficiently decomposed (*SD*).

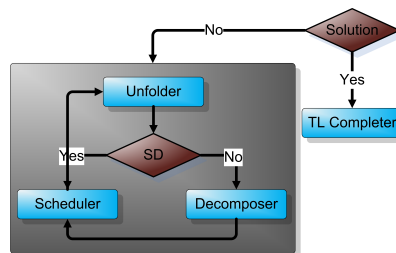


Figure 4.6.: Next resolver strategy - UDS (Unfolder, Decomposer, Scheduler).

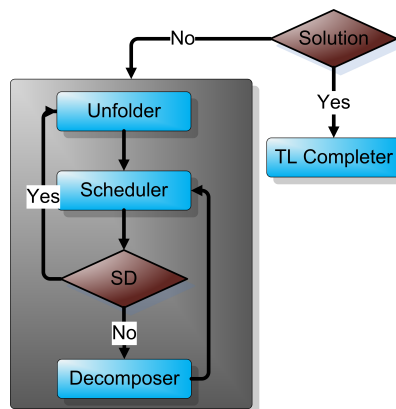


Figure 4.7.: Next resolver strategy - USDS (Unfolder, Scheduler, Decomposer, Scheduler).

Algorithm 3 illustrates how the outputs produced by the resolvers are analysed.

Algorithm 3: Supersolver.analyseOutput()

```

begin
  while (tasksFinished(sharedInfo) > 0) do
     $\rho \leftarrow \text{getNextFinished}(\text{sharedInfo})$ 
    subtask  $\leftarrow \text{getProblem}(\rho)$ 
    plans  $\leftarrow \text{getSolutions}(\rho)$ 
    freeResolver(sharedInfo,  $\rho$ )
    if (isDeadEnd(subtask)) then
       $\text{closedList} \leftarrow \text{subtask}$ 
    else if (solutionType(subtask) = RESOLVER_STEP) then
       $\text{openList} \leftarrow \text{subtask}$ 
    else if (solutionType(subtask) = PARTIAL_SOLUTION) then
       $\text{openList} \leftarrow \text{extractNewProblem}(\text{subtasks})$ 
    else if (solutionType(subtask) = SOLUTION) then
       $\text{solutions} \leftarrow \text{subtask}$ 

```

The loop iterates through all the jobs finished at the time the function is called. First, the problem (a subtask represented as a node n_i^{sub} containing a dn) and its solutions (a set of plans Π_i equally represented as nodes) are retrieved.

If there is no solution, then the subtask and all its supertasks (if any) are labelled as *DEAD_END* and stored in the closed-list. Otherwise, each plan is analysed according to its solution type, which is defined by the resolver.

Example 14 Continuing with the search space showed in Figure 4.4, Figure 4.8 illustrates how a *DEAD_END* node affects the open-list. Suppose that the subtask n_{11} finds no solution. In consequence, its parents n_1 and n will also be labelled as *DEAD_END* because it is not possible to find a solution for the supertask if one of the subtasks has none. Moreover, the rest of children of n_1 and n will also be labelled as *DEAD_END* as it is already known that the supertask does not have any solution.

If the k -th plan $\pi_{ik} \in \Pi_i$ is a *RESOLVER_STEP*, other resolvers will be still required, therefore π is stored in the open-list.

In case it is a *PARTIAL_SOLUTION*, no further resolver is needed but π_{ik} needs to be added to the rest of subplans to create a whole solution. The Supersolver checks then whether all the subtasks n_i^{sub} of the supertask n^{super} have been resolved. If it is so, then *extractNewProblem*(*subtasks*) combines all the subtasks into a new node that will be added to the open-list. This new node will require at least an extra scheduling step before becoming a solution. In case some subtasks are still pending to be resolved, π is stored in a hash which key is the supertask. Finally, if the type is *SOLUTION*, it means that n_i^{sub} has no supertask and therefore the plan is stored in the list of solutions.

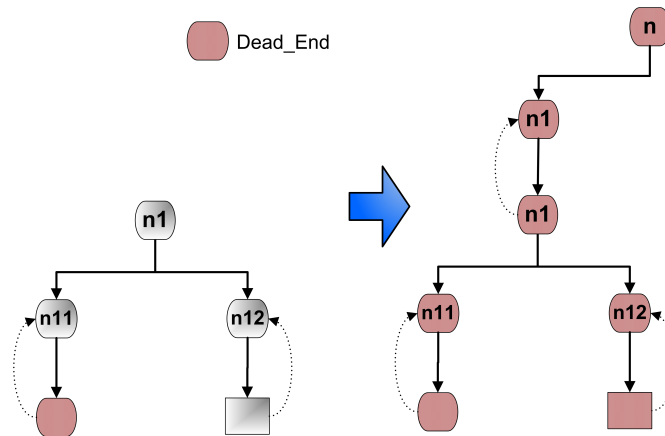


Figure 4.8.: Propagation of a *DEAD_END* subtask up to the supertasks.

Problem division and recombination

Given a problem P containing an initial state st_{init} formed by a list of facts (typically one per timeline) and a totally ordered goal state containing zero or more goals for each timeline, the maximum number of subproblems in which P can be divided is equivalent to the maximum number of goals max_goals of any timeline in the goal state. The process is better explained with an example.

Example 15 Figure 4.9 presents a problem for a simplified version of the Primary Rover in the FASTER scenario. In this case, the system is composed of three timelines (Locomotion, Path Planner and Antenna), where the yellow boxes represent the initial facts and the white ones goals.

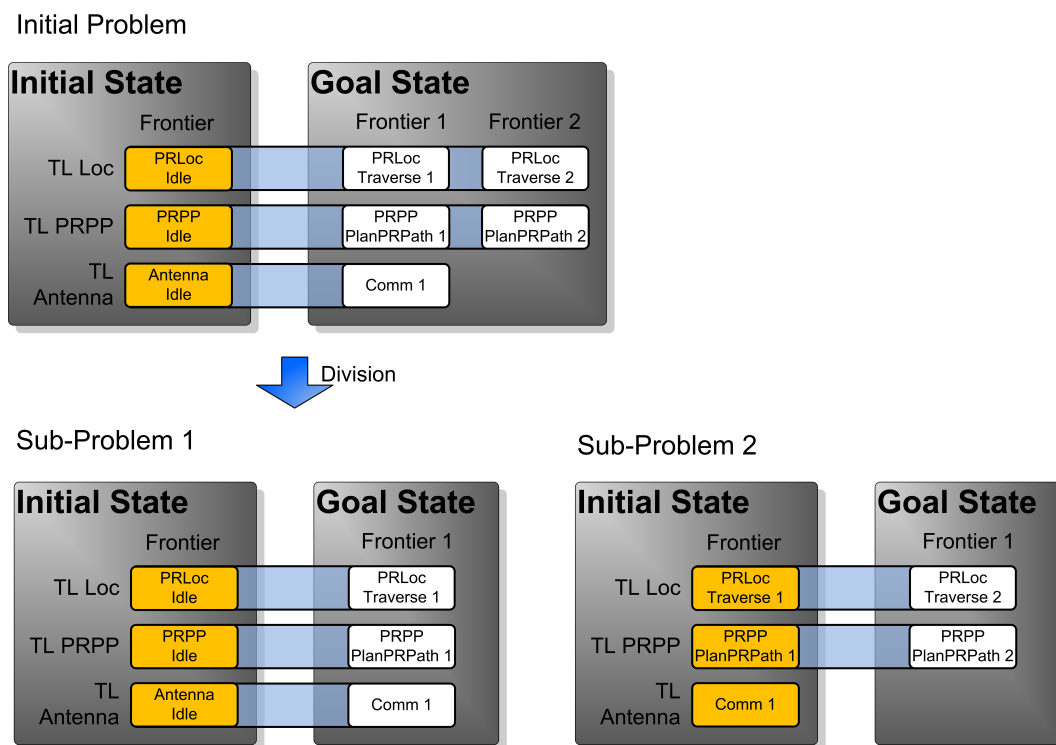


Figure 4.9.: Problem division in subproblems.

The timelines *TLLoc* (Locomotion) and *TLPRPP* (Path Planning) have the maximum number of goals with a total of two, which is the value assigned to *max_goals*. In consequence, two frontiers are generated for the goal state: $Frontier_1 = \{Traverse_1, PlanPRPath_1, Comm_1\}$ and $Frontier_2 = \{Traverse_2, PlanPRPath_2\}$.

The first subproblem P_1 has as an initial state all the facts in st_{init} while the goal state contains the first frontier, $Frontier_1$. Subsequent subproblem P_i has as facts the goals of P_{i-1} and as goals the next frontier of the goal state, in this case $Frontier_2$. The process iterates *max_goals* times until the last goal state frontier is reached.

The unification is simpler. Given *max_goals* subproblems, the Supersolver will try to solve each of them. In case it is not possible to find a solution for one P_i , st_{init} is declared as *DEAD_END*. Otherwise, for each subproblem the Supersolver receives a list of one or more subplans and computes the Cartesian product of all of them. The resulting combination represents the list of children of st_{init} .

4.3.3 Resolvers

The resolvers are tactical algorithms in contraposition to the strategic approach of the Supersolver. Given a *SuperSearchNode*, a resolver tries to evolve the initial state to reach the goal state. For example, in the case of the *unfolder*, given an initial decision for a *StateVariable* component, it tries to find the path to the goal in the automaton defined by the component, adding the most promising successors to the *DecisionNetwork*.

A resolver receives as inputs a *SuperSearchNode* with an initial and goal state, a number of configuration parameters specific to the resolver, a number of parameters for synchronization purposes with other resolvers running in parallel, a list of heuristics (one or more) to evaluate the *h*-value of a node, a meta-heuristic to choose which heuristic to use and a *g*-evaluation function to determine the cost of the node. It returns to the Supersolver a list of solutions sorted according to the *f*-value ($g + h$) of each node.

Unfolder

The *unfolder* is the equivalent of the node expansion algorithm in classical planning. It is the responsible for one of the main novelties of the planner, the forward-chaining in the state-space search strategy in opposition to the plan-space iterative repair approach followed by most (if not all) TLP planners such as *AP²*. As explained in Sections 2.3.1 and 2.4.2, this approach has demonstrated to be the most successful for classical planning. Its benefits (presented in Section 2.1) with respect to Partial Order Planning (POP), can be summarized as: (1) More informed heuristics thanks to the whole definition of the current state; (2) No need to resolve threats thanks to the early ordering of the decisions. Forward-chaining is preferred to backward-chaining due to the smaller search tree generated (see Section 2.3.1).

In spite of all the benefits, constructing a linear *unfolder* presents several difficulties that needed to be addressed to make it efficient. While most of the problems cited in Section 2.1 were solved in *QEv2*, the non-linear nature of temporal problems with parallel actions represents a big challenge that will be further analysed at the end of this section.

Forward-chaining is just a strategy that can use any of the algorithms presented in Section 2.4.1. *QuijoteExpress*' *unfolder* is based on *A** because of its nice properties: it is complete and optimally efficient when used with consistent heuristics. As the space complexity of *A** is worst than its time complexity, a variant called *SMA* (Simplified Memory-bounded *A**) can be used in case computing resources are scarce, for example when executed in the on-board computer of a robot.

When the unfolder is called by the Supersolver, it receives a number of inputs (listed above) including an initial and goal nodes. The initial node is stored in the open-list of the search space and search begins as shown in Algorithm 4.

Algorithm 4: `unfolder.call()`

```

begin
  while ( $\neg exitCond()$ ) do
     $node \leftarrow selectNextNode(openList)$ 
    if ( $hasNode(closedList, node)$ ) then
       $continue$ 
    if ( $isSolution(node)$ ) then
      if ( $Frontiers_{gs} == 1$ ) then
         $solutionType(node) \leftarrow PARTIAL\_SOLUTION$ 
      else if ( $Frontiers_{gs} > 1$ ) then
         $solutionType(node) \leftarrow RESOLVER\_STEP$ 
       $solutions \leftarrow node$ 
    else
       $successors \leftarrow getSuccessors(node, goal)$ 
      if ( $successors = \emptyset$ ) then
         $solutionType(node) \leftarrow DEAD\_END$ 
      else
         $solutionType(node) \leftarrow RESOLVER\_STEP$ 
         $solutions \leftarrow successors$ 
         $closedList \leftarrow node$ 

```

First of all, the algorithm checks the exit conditions, which are equal to those of the Supersolver described in Equation 4.1.

In case no exit condition is met, the algorithm gets the next node of the open-list (typically the one with lower cost), which becomes the current node $n_{current}$ and checks by means of $isSolution(node)$ whether $n_{current}$ is already a solution. The current/initial state $st_{current}$ associated to a node n is a solution if the goal state st_{goal} of the node is empty. In this case, the node is labelled as *PARTIAL_SOLUTION* and stored in the list of solutions.

If the current node $n_{current}$ is not a solution, the list of successors from $st_{current}$ towards st_{goal} is extracted. If there are no successors, $n_{current}$ is a *DEAD_END*. Otherwise, the successors are stored as *RESOLVER_STEPS* in the solutions list and the algorithm returns.

Example 16 *Continuing with the Primary Rover scenario presented in the previous example, Figure 4.10 shows that the current state³ has been expanded, being its latest frontier $Frontier_{current} = \{Traverse, PlanPRPath, Comm\}$. On the other hand, the earliest frontier of st_{goal} is $Frontier_{gs_1} = \{Traverse_1, PlanPRPath_1, Comm_1\}$. In case $CanUnify(Traverse, Traverse_1) = \top \wedge CanUnify(PlanPRPath, PlanPRPath_1) = \top \wedge CanUnify(Comm, Comm_1) = \top$, the expanded current state is a solution for $Frontier_{gs_1}$. As st_{goal} still has more frontiers, $st_{current}$ solving type is labelled as *RESOLVER_STEP*.*

Regarding $getSuccessors(node, goal)$, given the current state $st_{current}$, the algorithm gets first a list of all the timelines having a goal in the goal state st_{goal} . For each of these timelines tl , the latest

³ At the beginning, initial and current states are equal

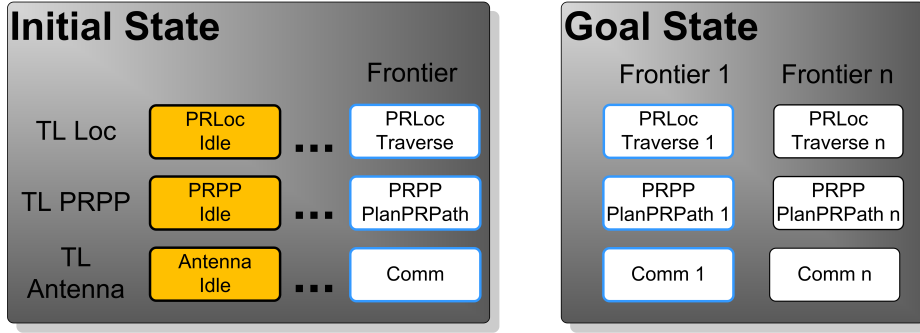


Figure 4.10.: Example of a matching frontier.

frontier in $st_{current}$ named d_{ifront}^{tl} and the earliest frontier in st_{goal} referred to as $d_{gfront_1}^{tl}$ are extracted. These two values correspond to the initial and goal state for which the method must find a path in the associated component automaton. The unfolders use an A* algorithm to search an optimal path until one of the following conditions is met:

- If the current successor d_{succ}^{tl} is unifiable with the decision of the timeline frontier in the goal state $d_{gfront_1}^{tl}$ (see Section 4.2.2), then the goal has been reached.
- The whole search space is exhausted and the goal is not reached.
- A non-deterministic decision is reached. A successor d_{succ}^{tl} of the decision d_{pred} is considered non-deterministic if: (1) The number of successors of d_{pred}^{tl} is larger than one; (2) d_{succ}^{tl} is a complex decision or the reference of a synchronization.

Having generated for each timeline different lists of successors d_{succ-i}^{tl} , one for each possibility of the non-deterministic step, the Cartesian product of the lists of successors of the different timelines will represent the list of solutions found.

Example 17 Given the initial and goal states shown in Figure 4.11, the unfolders need to search successors just for the first two timelines (Locomotion and Path Planning), as the third (Antenna) does not have goals.

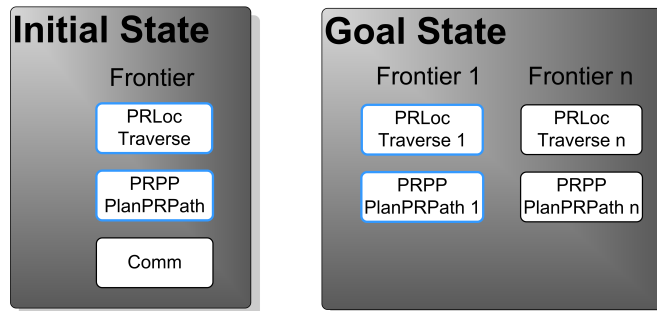


Figure 4.11.: Selection of timelines for which the unfolders need to search successors.

In the left hand of Figure 4.12, the automaton for the component Locomotion is represented. The initial state frontier d_{ifront}^{Loc} is labelled as I and the earliest goal frontier $d_{gfront_1}^{Loc}$ as G . The algorithm returns the two possible paths connecting I and G ordered according to their cost, which has been heuristically evaluated. For example, if each edge has a unitary cost, the first path costs 3 units while the second costs 2.

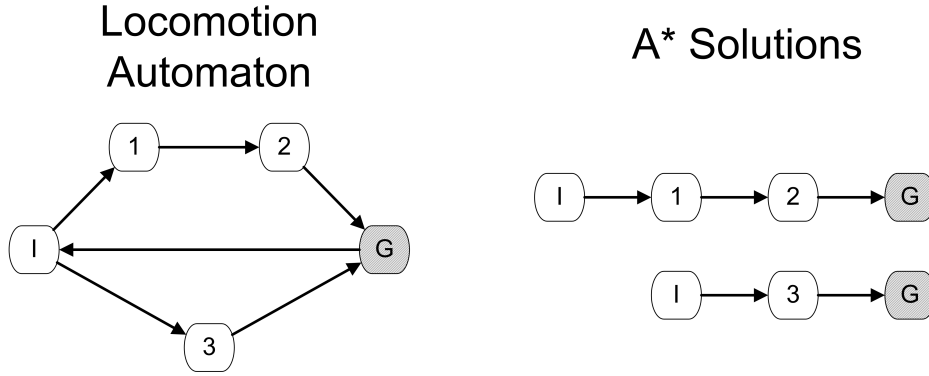


Figure 4.12.: Path search based on an A* algorithm for a StateVariable component.

The list of successors will be extracted by combining these two solutions with the solutions obtained for the PathPlanning timeline.

Once the concepts behind the unfolders have been presented, it is possible to explain the problem derived from the (non-linear) nature of the dependencies generated by the decompositions and synchronizations. When a resolver ρ_δ decomposes a complex decision d^c in a subnetwork dn_d , the subnetwork will typically contain decisions and constraints belonging to different components. In order to make the planner complete, the scheduler must search all possible orderings for each decision $d^{tl} \in dn_d$. In case d^{tl} can be placed between the initial decision d_{init}^{tl} and the frontier of the timeline d_{finit}^{tl} , all the decisions between d^{tl} and the initial frontier will not be valid anymore. As a consequence, the heuristics cannot be admissible as the h - value representing the distance to the goal will necessarily increase. Therefore, A* will not be optimal between different runs in terms of node expansions.

Decomposer

The decomposer is the equivalent of the task reduction algorithm in HTN. This resolver is the responsible for the second novelty of this planner which is the decomposition of complex decisions in subdecisions according to the HTLN formalism.

With the inclusion of HTN capabilities in a TLP planner, the planning process can be analysed from two different dimensions:

- Horizontal (Temporal): Ordering of the decisions of the plan in the timeline.
- Vertical (Granularity): Decomposition of the decisions up to a certain level of detail.

The horizontal dimension is well defined by the interval $H = [t_0, t_H)$ from the starting point to the horizon. However, the level of granularity is not that clear. The planner can only plan down to the maximum level of detail at which the model has been defined. The executive will be then in charge of translating the actions of the plan to the appropriate low-level functions executable by the different subsystems. Planning to the lowest level is often undesirable because of the resulting combinatorics of deep search. Furthermore, planning should be limited down to abstraction levels at which actions can be reasoned ahead on time.

When the decomposer ρ_δ is called by the Supersolver, it receives a number of inputs including an initial and goal nodes. The initial state, which contains at least one complex decision d_{vc} , is stored in the open-list of the search space and the decomposition process starts as described in Algorithm 5.

Algorithm 5: decomposer.call()

```
begin
  while ( $\neg$ exitCond()) do
    node  $\leftarrow$  selectNextNode(openList)
    if (hasNode(closedList, node)) then
       $\lfloor$  continue
    if (isSolution(node)) then
       $\lfloor$  solutionType(node)  $\leftarrow$  PARTIAL_SOLUTION
       $\lfloor$  solutions  $\leftarrow$  node
    else
       $\lfloor$  successors  $\leftarrow$  decompose(node)
      if (successors =  $\emptyset$ ) then
         $\lfloor$  solutionType(node)  $\leftarrow$  DEAD_END
      else
         $\lfloor$  solutionType(node)  $\leftarrow$  RESOLVER_STEP
         $\lfloor$  solutions  $\leftarrow$  successors
         $\lfloor$  closedList  $\leftarrow$  node
```

The main structure of the algorithm is the same as for the unfold. In the decomposer however, the function *isSolution* is different as it uses *decompose(node)* instead of *getSuccessors(node, goal)*.

isSolution returns true in case the level of decomposition of the DecisionNetwork in the state $st_{current}$ associated to the node $n_{current}$ is equal or higher than the minimum level of decomposition defined in the configuration file, which can be *PARTIAL* or *FULL*⁴

The function *decompose(node)* follows the principles defined for the Decomposition resolver ρ_δ in Section 3.2.12. It is based on a recursive approach that alternates two functions, *decomposeNet* and *decomposeDec*, in charge of decomposing a decision network and complex goal respectively. The first one just checks if the network is *NOT_SUFFICIENTLY* decomposed, in which case it calls the function *decomposeDec* for each of the complex decisions d_{vc} in the network. In case d_{vc} cannot be unified with any method $m \in M$, the node is a *DEAD_END*. Otherwise, there are two possibilities: (1) There is one unifiable method: The step is deterministic, therefore the instantiated network θdn_d is generated and further decomposed; (2) There is more than one unifiable method: The step is non-deterministic, therefore an instantiated network θdn_{di} for each method m_i is generated (later on used to generate the Cartesian product) and the process finishes. The algorithm is sound and complete as demonstrated in Section 3.3.1. The pseudocode is represented in Algorithm 6.

As explained in the previous section, the subnetwork dn_d obtained from a decomposition or synchronization does not need to have a linear ordering. Therefore, the frontiers need to be recalculated in the scheduling step, once the scheduler has ordered all decisions of dn_d in the problem network dn .

Scheduler

QuijoteExpress uses a modification of the scheduler of AP², based on the algorithms presented in [37] which is sound but not complete.

⁴ Obviously, a *DN* which is *NOT_SUFFICIENTLY* decomposed does not represent a solution.

Algorithm 6: decomposer.decomposeDec()

```
begin
  if (getDecLevel(decision) == NOT_SUFFICIENTLY) then
    methods ← getDecompositions(Domain)
    if (methods = ∅) then
      solutionType(node) ← DEAD_END
      closedList ← node
    else
      if (|methods| = 1) then
        decomposeNet( $\theta dn_d$ )
      else
        decomposeNet( $\theta dn_{di}$ ),  $\forall dn_{di} = \theta(d_{vc}, m_i)$ 
```

The scheduler has been adapted to be thread-safe by means of a wrapper that implements a solving function similar to those showed for the unfold and decomposer. This modification allows the super-solver to launch the scheduler in an independent thread which might be running in parallel with other resolvers. In addition, it has been configured to order the decisions of the current and goal states and update the frontiers.

Timeline Completer

As a side effect of the forward-chaining approach followed by the unfold, no gaps are left in the timeline completer. In consequence, its duties are reduced to extract the timelines, which represent a particular solution of the multiple allowed by the decision network and the structure used by SanchoExpress for execution.

4.3.4 Heuristics

The change of paradigm from plan-space to state-space provides some advantages from the point of view of the heuristics:

- Allows to incorporate classical heuristics such as Landmarks or PDBs.
- The states are fully defined which makes easier to create more informed heuristics.

The heuristics should fulfil the following properties in order to be useful in the scenarios of interest identified in Chapter 1:

- (Mandatory) Safe Pruning: It is a necessary condition to achieve completeness. Non-safe heuristics might be required for certain problems, in which case it should be clearly stated.
- (Mandatory for optimal planning) Strongly Safe Pruning: In case the concept of optimality is defined for the problem at hand, for example by means of a function that evaluates the cost of each node, then strongly safe heuristics must be provided.
- (Mandatory for optimal planning) Admissible heuristics: Admissible heuristics guarantee that the search process expands the minimum number of nodes possible and that the best solution obtained

is the optimal one. It might be hard to achieve if the concept of distance to the goal or cost is not well defined.

QEv2 requires two groups of heuristics: MetaHeuristics, in charge of selecting other heuristics and heuristics to select the next node to analyse. An appropriate selection of the heuristics is fundamental, as they have a great impact on the performance of the planner. For example, in the IPC many planners which shared the same core but used different heuristics such as the family of FastDownward and Lama, showed important differences. Implementation of advanced heuristics represents a complex task outside the scope of this thesis. However, APSI* has been developed having in mind the integration of modern heuristics to the planner in the future.

MetaHeuristic

In case a multi-queue approach (see Section 4.3.5) is used to evaluate a node with several heuristics, some kind of policy, here called meta-heuristic, is required to choose the queue from which the next node to be expanded should be extracted. This schema can be further modified by assigning numerical priorities to each queue and using some queues more often than others [129].

The combination of the meta-heuristic and multi-queue are intended to exploit the strengths of different heuristics to improve the planner performance. This research field has not been explored in depth yet, even for classical planning. However, different meta-heuristics have been suggested such as those based on arithmetic operators (maximum, addition, etc.), Pareto-optimal or Alternation [131]. The meta-heuristics provided with QE are:

- mh^{max} : Selects among all the queues the node having the maximum h -value (see Section 2.4.1). When used with admissible heuristics, the resulting heuristic dominates all individual ones and usually requires fewer state evaluations to solve a task, having shown a very good performance in optimal planning.
- mh^{alt} : Alternates sequentially between the different queues. It has shown good performance in satisficing planning.

Next Node Heuristic

The development of heuristic estimators for optimal/satisficing classical domain-independent planning represents one of the most active research fields in planning. Heuristics can be classified into one of the following four groups: delete relaxations, critical paths, abstractions, and most recently, landmarks [87].

The scenario looks radically different in applied planning. While generic heuristics as those mentioned above are not so relevant, the use of knowledge to control search has been consistently used with great success as indicated in Section 2.1, which in turn has not received that much attention from the classical community, in part because it is specific to each problem and does not represent a good, general approach to compare different algorithms from a research point of view. However, this situation has changed with the appearance of landmark heuristics [92, 97, 87]. Landmarks present some similarities to HTN methods, as both contain information about elements (whatever an element is in the corresponding formalism) that a given plan should satisfy to achieve some goal.

The best strategy used by QE to prune the search space is actually not a heuristic, but the use of HTN methods [170]. Besides, two heuristics have been conceived for the unfold in order to choose the next node to expand:

- Least constrained value: The heuristic value assigned to the node corresponds to the addition of the minimal number of assertions among all the synchronizations and decompositions of the domain

for which each decision of the DecisionNetwork is a reference. The heuristic is not admissible, as it disregards the fact that some of these assertions might be already satisfied in the current network.

- Temporal distance: Estimates the minimal temporal distance from each decision in the frontier of the current node to the first frontier of the goal node according to the values precomputed for each component by the preprocessor. This heuristic is admissible.

4.3.5 Search Enhancements

Parallel Planning

Running several algorithms in parallel represents an important boost in the performance of any system. Three sources of parallelism have been identified in QE: (1) Dividing the problem according to the temporal organization of the goals in frontiers; (2) Run in parallel resolvers for different child nodes and multiple heuristic evaluations for a given node; (3) Dividing the problem according to its graph connected components.

From these three options, the first and third imply the division of the plan into subplans with the corresponding overhead related to the synchronization and further recombination of the subparts. In the specific case of APSI there is one additional problem: APSI has one single STN, which is stored in the DomainManager, forcing the scheduler to compute the whole STN for every new node. In consequence, two schedulers cannot be used at the same time in APSI (Section 4.4 present some ideas to mitigate this problem). The second type of parallelism could be used at no extra-cost because there is no dependency between different child nodes or between different heuristic evaluations for the same node. QE exploits the first two options.

Before the planning phase starts, the Supersolver attempts to divide the problem into subproblems with respect to the frontiers (see Section 4.2.2 for its definition and 4.3.3 for its extraction process), which are then resolved in parallel. Even though this process has been limited so far, the division of a problem could be further exploited inside the main loop of the Supersolver after every execution of the unfold or decomposer. If the result is optimal for every subnetwork dn^{sub} , the overall solution will also be optimal [170].

With respect to the second type, during planning the Supersolver is able to launch a new resolver for each successor obtained from the expansion of a node for each of the subproblems (in case the problem is divided). Moreover, it can launch in parallel several heuristics to evaluate a node in case the multi-queue approach is selected. Suppose a problem has been divided into m subproblems, that the average branching factor for a node is b and that the number of next-node heuristics is n . QE could be able in its most powerful configuration to run $m \times b \times n$ parallel threads.

Sufficient Planning

The traditional definition of complete, valid plan requires a full specification and ordering of all the activities, with all the variables grounded and all constraints satisfied. However, this is an unnecessarily restrictive definition. In many situations with high levels of uncertainty such as those faced by rescue robots or Mars rovers (see the example in Section 4.2.1), the information required to define an activity that lies inside the planning horizon might still not be available, yet it is required to generate a valid plan.

In other cases, it might be desirable to allow the executive to complete the unspecified parts of the plan (see the issues related to plan granularity in Section 4.3.3).

To solve this problem the concept of sufficient planning, theoretically described in Section 3.3.2, has been introduced with the aim to handle uncertainty and create more robust plans by decreasing the number of assumptions needed to be taken at the moment the plan is first generated.

QE achieves sufficient planning thanks to the combination of HTLN (more specifically the concept of complex and primitive values) and some additional information added to the desired complex goals in the problem description by means of the label *ND*. This label is used to tell the decomposer which decisions do not need to be further decomposed as previously explained in Section 4.2.1.

Deferred Heuristic Evaluation

Upon expansion of a state s , informed algorithms such as Greedy best-first or A* compute the heuristic evaluation of all s successors and sorts them into the open-list. This can be wasteful if the search space presents a big branching factor and the evaluation of the heuristics is computationally expensive, two conditions that are often true for heuristic search approaches to planning.

With deferred heuristic evaluation, once a state s is removed from the open-list for expansion, its heuristic value is calculated and assigned to all its successors with the resulting decrease in the number of heuristic evaluations. However, this technique usually increases the node expansions and therefore, has higher space requirements compared to standard best-first search because the heuristics are less informative. Nonetheless, heuristic evaluations are usually so costly in time (they make up 80% of the runtime in Fast Downward [129]) that memory is not a limiting factor.

This technique, borrowed from the FastDownward planner [86], has demonstrated that it can improve search performance, especially in problems where branching factors are large and the heuristic estimate is informative [126]. QE applies deferred heuristic evaluation in both the Supersolver and resolver.

Multi-queue

This technique has been also inspired in classical planning [86, 131, 128].

Nowadays a wide range of heuristics is available, none of which consistently outperforms the others. In this approach, the open-list is divided in several queues, each associated to one heuristic. Each new state is evaluated with respect to all heuristics and added to all the open-list queues, in each case with its corresponding heuristic value. When a new node is required for expansion, the search algorithm selects the most promising node (the head) from the queue selected by the meta-heuristic, deleting the node from the rest of queues.

This approach is ideal for QE as it can evaluate all heuristics at virtually no extra temporal cost thanks to its parallel approach. Because the evaluation of each heuristic is independent from the rest, there is no overhead due to synchronization activities as it happens in the division of a problem and recombination of subproblems.

Moreover, in case the heuristics used are computationally expensive and parallel planning is not used, another approach of QE called **Single Evaluation Multi Heuristic** offers the possibility to alternatively evaluate just one of the heuristics, which selection is also based on the meta-heuristic, in which case the open-list contains just one queue.

4.4 Evaluation

4.4.1 Testing QEv1

In [167], the performance of QuijoteExpress based on a flaw-repair approach and AP² planners were compared against two synthetic rover domains (see Table 4.3 for a comparison of their properties).

The first domain, called *RD-C* (Rover Domain Complex) fully represents the model presented in Figures 4.13 and 4.14. The second one named *RD-S* (Rover Domain Simple) has removed a level of decompositions, as *Driving* is directly decomposed in the primitive tasks without the *Blind* and *Autonav* intermediate layer. Moreover, the *Navigation* and *Driller* components have been removed. Further details are provided in [167]. Each of these models have been defined in a hierarchical and flattened style in order to compare QEv2 and AP². The modifications included the translation of the decompositions as synchronizations, which have similar semantics, while no additional constraints have been required.

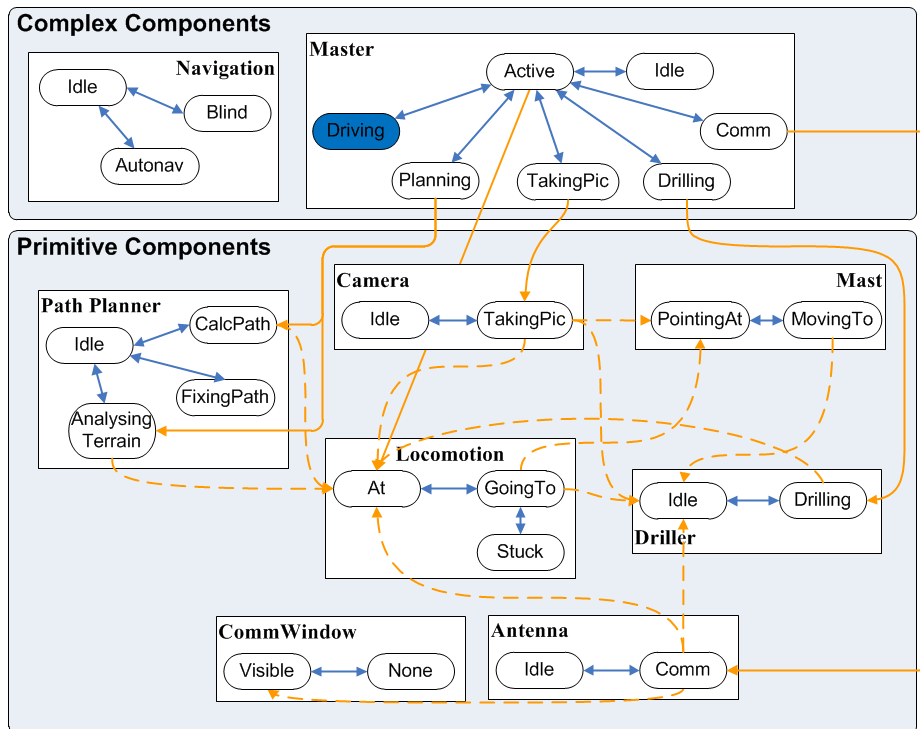


Figure 4.13.: Hierarchical rover model.

Property	RD-C	RD-S
Number of timelines	9	7
Total number states	27	19
Number decompositions	4	2
Number synchronizations	14	6

Table 4.3.: Properties of synthetic rover domains for QEv1.

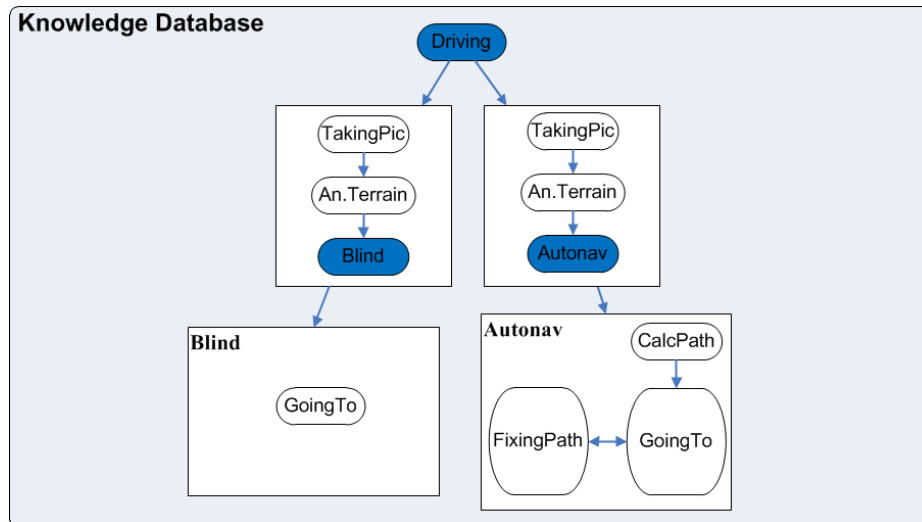


Figure 4.14.: Knowledge database associated to the synthetic rover domains to evaluate QEv1.

A total of 7 problems named *Problem-1* to *Problem-7* with increasing level of complexity were created. Each problem *Problem-x* contains a list of *facts* defining the initial state for each component and x high-level goals, i.e. goals from the *Master* component. A simplified instantiation of *Problem-4* for the domain *RD-C* is shown below.

```

PROBLEM Rover-Problem (DOMAIN Rover_Domain){
  f1 <fact> Locomotion.At(?x=0, ?y=0);
  f2 <fact> Mast.PointingAt(?pan=0, ?tilt=0);
  f3 <fact> Camera.CamIdle();
  f4 <fact> Driller.DrillIdle();
  f5 <fact> Planner.PlannerIdle();
  f6 <fact> Antenna.CommIdle();
  f7 <fact> Navigation.NavIdle();
  f8 <fact> Master.Idle();

  f9 <fact> CommWindow.Visible() AT [0,5];
  f10 <fact> CommWindow.Visible() AT [40,80];

  g1 <goal> Master.Drive(?x=1, ?y=1, ?trav=easy);
  g2 <goal> Master.Pic(?x=1,?y=1,?pan=2,?tilt=2);
  g3 <goal> Master.Drill(?x=1, ?y=1, ?depth=1);
  g4 <goal> Master.Drive(?x=4, ?y=4, ?trav=hard);

  g1 BEFORE [1,+INF] g2;
  g2 BEFORE [1,+INF] g3;
  g3 BEFORE [1,+INF] g4;

```

Most of the parameters in the decisions of the domain are self-explanatory with the exception of *?trav* in the goal *Drive*, which describes the type of terrain and is used at planning time to decide the type of navigation the rover should do. Few modifications were required to create similar problems for the *RD-S* domain. *Drilling* activities are replaced by *TakingPicture* and facts *f4* and *f6* were removed from the initial state.

Both planners have been configured to search for all possible solutions. Completeness can be achieved by the unfold and decomposer, but it is not possible for the scheduler provided by APSI. Additionally, QE was configured to run in single thread mode, with single heuristic evaluation (blind heuristic) and with *FULLY* decomposition mode, that is, even those goals indicated to be partially decomposed in the

problem would be in the end finally decomposed, increasing the work load not only for the decomposer, but also for the unfold and scheduler.

The tests have been run in an Intel Core i5 M540 at 2.53 GHz computer with 3 GB RAM and Windows 7 Enterprise 32 bits. To understand the level of impact of the different decomposition methods in the performance, both QEv1 and AP² were configured to choose randomly the decomposition/synchronizations to apply. Notice that for the rover domain, the decomposition of a *Driving* goal as *Autonav* impose way more constraints than *Blind*.

For each run, several parameters were recorded, including the time required to find the first solution and the number of nodes/edges in the solution network. The results comparing QEv1 and AP² are shown in Figures 4.15-4.18.

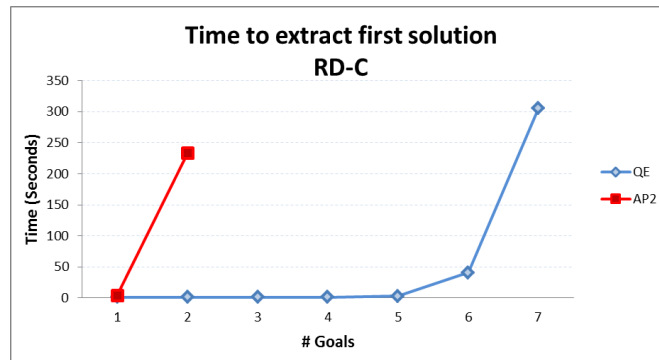


Figure 4.15.: Performance of QEv1 and AP2 in *RD - C*.

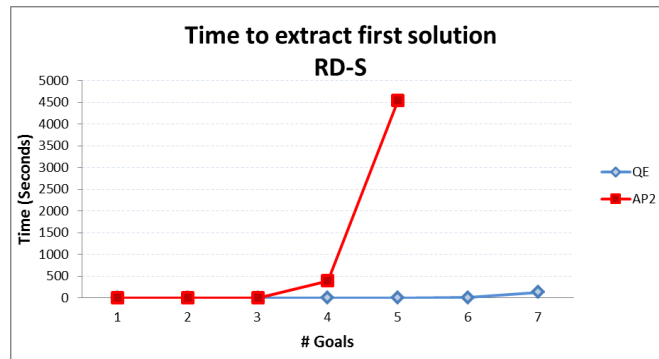


Figure 4.16.: Performance of QEv1 and AP2 in *RD - S*.

In the first domain, *RD-C*, AP² starts to have problems to find solutions even with very simple problems due to the big branching factor experienced during scheduling. AP² has to branch for all possible disjunctive synchronizations and then unfold supporting actions and schedule them. On the other hand, QEv1 uses the decomposer to find an appropriate method and directly insert it as a sub-plan in the decision network, decreasing the number of calls to the unfold and specially to the scheduler.

In the second scenario, AP² managed to solve up to *Problem-5*. As the problem complexity grows, the number of edges increments notoriously faster than the number of nodes. In addition, QEv1 consistently generate less edges than AP². These two facts are crucial to understand the degradation in AP² performance: AP² uses an all-pairs shortest path algorithm to propagate the temporal network which time complexity is quadratic to the number of time points when new constraints are added. It is also interesting to remark that the peak in time does not come with the second *Driving* activity, which im-

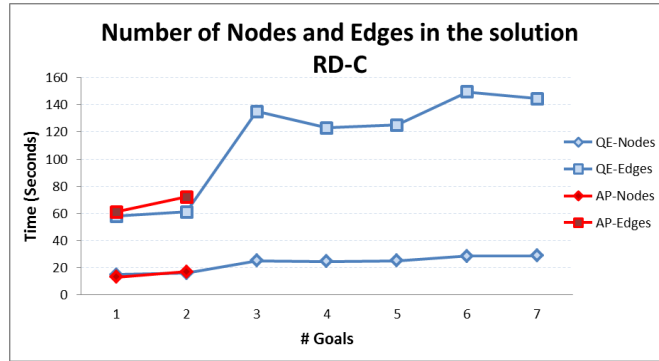


Figure 4.17.: Number of nodes generated by QEv1 and AP2 in $RD - C$.

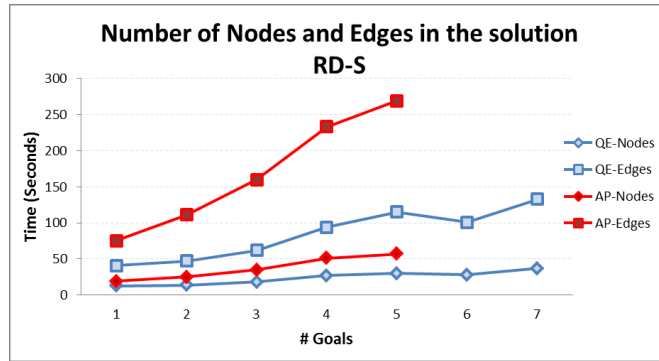


Figure 4.18.: Number of nodes generated by QEv1 and AP2 in $RD - S$.

poses itself a number of subtasks and subrelations, but with the next activity which is *TakingPicture*. The reason is that after the second *Driving*, the planners (unfolder and scheduler) need to do a lot of processing to insert *TakingPicture* and unify some of its parameters. The same effect can be seen in the first scenario with QEv1.

4.4.2 Testing QEv2

AP² has been also compared with the new version based on forward-chaining. A variant of $RD-S$ (see Table 4.4) with just three components and one single decomposition has been used to maximize the iteration between different resolvers.

The problems from the previous experiment have been adapted to the new model. Two sets with increasing complexity have been created: one having all goals fully ordered by f_{temp}^{before} constraints and the other without any ordering.

Property	RD-B
Number of timelines	2
Total number states	5
Number decompositions	2
Number synchronizations	0

Table 4.4.: Properties of the synthetic rover domain for QEv2.

The target of these tests is to evaluate the performance of the forward-chaining and parallel planning approach. Evaluation of other enhanced features such as deferred heuristic evaluation and multi-queue heuristic evaluation heavily depend on the heuristics used and fall outside the scope of this thesis. However, taking into consideration that the unfolders and decomposer follow the same approach than most classical planners, previous studies such as [87] or [128] could shed light on the positive impact of advanced heuristics on the performance of QuijoteExpress.

The planners have been configured to return the first solution. QuijoteExpress has been tested in single-thread (labelled *QE-ST* in the graphs) and multi-thread (labelled as *QE-MT*), in both cases with multiple blind heuristics running in a multi-queue configuration. Configuring QE in multi-threading also implies the division of the problem in subproblems, activity performed by the preprocessor prior to the process of planning. The search time was limited to 180 seconds and the level of decomposition forced to *FULLY*. The computer was the same used in the previous test.

Figures 4.19 and 4.20 presents the results of total order and partial order plans respectively.

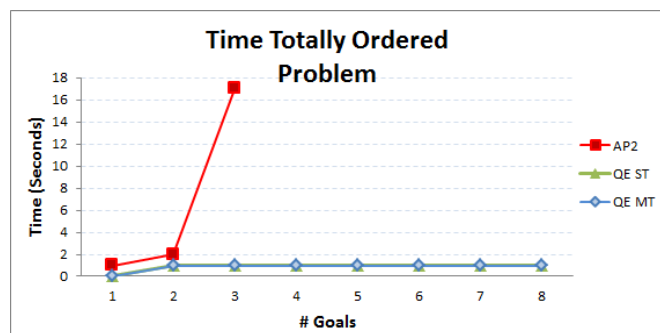


Figure 4.19.: Performance of QEv2 and AP² in Totally Ordered Problems.

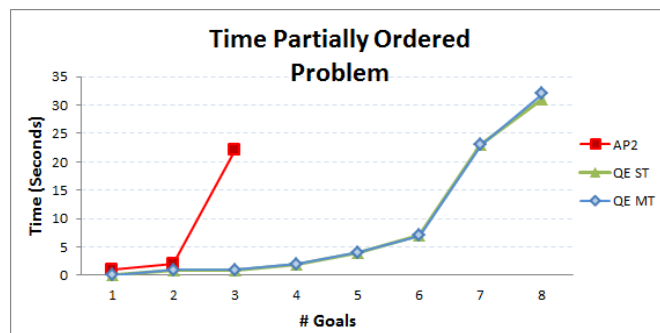


Figure 4.20.: Performance of QEv2 and AP² in Partially Ordered Problems.

As explained in the previous experiment, the difference in the performance of the planners with the two sets is due to the time required to propagate the temporal network. Providing a total order among the goals greatly restricts the possible solutions and therefore the effort required by the scheduler. Specially significant is the case of QE, which required almost constant time to solve the totally ordered problems but required a considerable time to solve problems 7 and 8 of the partially ordered set (1 second vs. 30 seconds in the last problem).

QEv2 shows better performance than QEv1, increasing the distance with respect to AP². The impact of multi-threading is negligible due to the low impact of the unfolders in the overall computational time as indicated by the statistics extracted with a profiler (see Figure 4.21). To understand the effect of multi-threading, a time delay of 100 milliseconds was placed in the function of the unfolded in charge

of generating the successors. The intention was to simulate a deeper search originated by complex automata. The percentage of CPU usage by the unfoldr climbed from 1.2% to 2.7% while the time required to solve *Problem* – 5 was 21 seconds for the single-thread version and 14 for the multi-thread. The benefit of multi-threading would be more relevant in highly complex scenarios, provided that APSI had a parallelizable scheduler (see Section 4.3.5). To overcome this limitation, the unfoldr and scheduler could be integrated taking advantage of the forward-chaining approach. To do that, the scheduler should be re-designed to perform incremental schedules as individual actions are sequentially added to each timeline by the unfoldr, strategy that has been already used by OPTIC, one of the best PDDL temporal planners available nowadays[10].

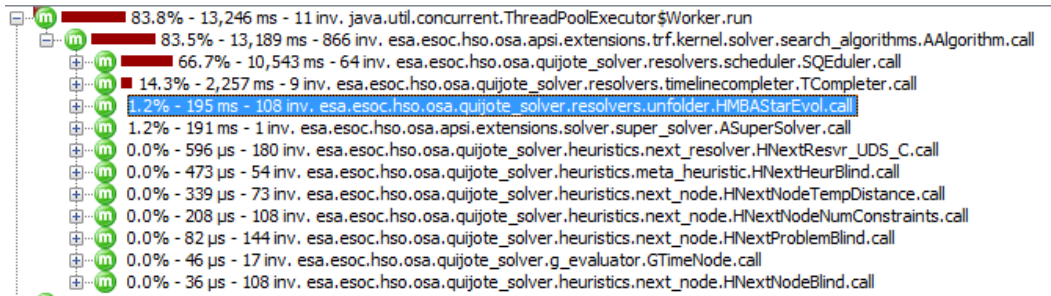


Figure 4.21.: CPU usage by QEv2 resolvers.

Another consequence of parallelism, more specifically from the division of the problem in sub-problems, is that the size of the search space decreases because the size of several small problems (the subtasks) is smaller than the size of a single big problem. This effect can be easily appreciated in Tables 4.5 and 4.6: QE-MT consistently creates less nodes and evaluates less heuristics than the single-threaded version.

The comparison of the complexity of the solutions generated from divided/non-divided problems is however inconclusive. On the one hand, divided problems consistently produce, as expected, solutions with less decisions because different goals in non-divided problems can use the same supporting activity while in a divided problem the supporting activity will have to be duplicated. On the other hand, no approach dominates in terms of number of edges, something that has not been fully understood and would require further research.

#Goals	#Solutions	#Nodes Created	#Heuristics	Decisions in Sol.	Edges in Sol.
3	2	50	101	7	9
4	3	162	330	9	19
5	4	412	842	11	26
6	5	890	1820	13	31
7	6	1710	3495	15	36
8	7	3010	6146	17	42
9	8	4952	10100	19	50
10	9	7722	15732	21	58

Table 4.5.: Results of QEv2-ST for Partial Ordered Problems.

#Goals	#Solutions	#Nodes Created	#Heuristics	Decisions in Sol.	Edges in Sol.
3	2	62	123	7	10
4	3	162	339	10	18
5	4	385	792	11	27
6	4	722	1507	14	32
7	5	1333	2814	16	37
8	5	2168	4695	18	43
9	7	2255	4750	20	47
10	4	2627	5499	24	54

Table 4.6.: Results of QEv2-MT for Partial Ordered Problems.

4.5 Conclusions

QuijoteExpress is a HTLN temporal planner that incorporates multiple novelties:

- Effective combination of forward-chaining, HTN and TLP planning techniques used in the unfold, decomposer and scheduler resolvers respectively.
- Use of sufficient planning to produce partial, valid plans in scenarios with high levels of uncertainty.
- Parallel planning capabilities to improve performance.
- The possibility of exploiting classical planning heuristics.

All these novelties make QuijoteExpress notoriously different from any other planner. In classical planning, POPF2 [50, 51] and specially OPTIC [10] incorporated temporal reasoning to a PPDL planner based on a POP approach, while ASPEN [42] is the only planner in use nowadays based on Timeline planning using HTN.

Even with poor heuristics, the results provided were better than expected, showing a big improvement with respect to previous APSI-based planners as shown in Section 4.4. The tests ran with the planner provided important lessons learnt:

- Better performance: The capability of HTN planners to reduce the branching factor and therefore the size of the search space is crucial to understand the great benefit in terms of performance.
- HTN represents an overhead during the modelling process: The design and validation/verification of the models implemented (hierarchical and non-hierarchical) was tedious and error prone. This fact is particularly relevant in HTN, where the model can present hidden constraints/dependencies between elements in different levels of abstractions. In consequence, it will be crucial in the future to develop new tools able to assist during the construction of complex models.
- Debugging and understanding the output of the planner is very difficult for a non-expert: HTN plays an important role in these two aspects. During the runs with the hierarchical model, it was possible to identify failing conditions just by observing the high-level goals and their relations. It would have taken much more time in case a flat model had being used. In the same way, it is easier for the user to understand the plan by examining only the timelines related with complex components.

-
- Modern domain description languages are not expressive enough: Even temporal-oriented modelling languages such as DDL impose several restrictions. Extending expressiveness or moving towards the use of standard programming languages, as it happens with SMACH for ROS, will be required to construct realistic models.
 - It is important to provide tools that allow users to represent knowledge: Domain-dependent knowledge (HTN methods in the case of QuijoteExpress) must be contained in dedicated structures to improve the system reusability.
 - General vs. dedicated heuristics: The use of general heuristics such as method timespan, number of complex tasks, etc. are not appropriate because the selection depends on the same specific knowledge contained in the method. For example, in the FASTER domain, the selection between the different type of traverses rely on the knowledge represented in the methods.

5 SanchoExpress: Flexible Execution with FDIR Capabilities

This chapter focuses on the reactive part of the architecture, more specifically on SanchoExpress (SE), an executive that performs dispatching and monitoring of the commands that constitute the plan calculated by QE. Besides SE, a robotic architecture named QEA (QuijoteExpress Architecture) was developed. QEA includes a number of components such as RobCon, platform-dependent executives, communication services, data types or visualization tools, some of which will be also presented. The following section introduces the 3-tier architecture where the planner, executive and rest of components are situated in place. Next, the different components (with the exception of the mission planner previously presented) are individually analysed and finally a number of conclusions related to plan execution are extracted.

5.1 Architecture

To cope with the need for continuous modifications/repairs of the plan intrinsic to robotic operations in the reference scenarios, the deliberative and reactive layers must be tightly coupled. Achieving it in QEA proved to be challenging due to low-level implementation issues between APSI and ROS. The plans produced by QE need to be translated to ROS messages and the other way around for execution and replanning respectively. The additional latency added to the system by this intermediate step is however negligible compared to the reaction time of the (re)planner.

The final design of QEA, as illustrated in Figure 5.1, is a 3-tier architecture where the planner and executive closely interact in an agile manner.

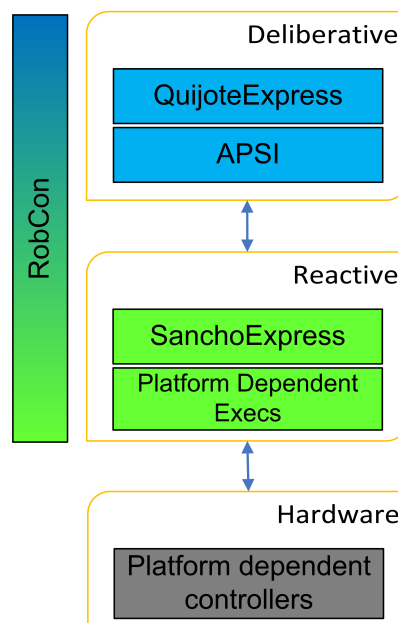


Figure 5.1.: Three layers architecture with QE and SE.

In the deliberative layer, QE is in charge of generating and repairing the plan. In the reactive, SE performs dispatching and monitoring while in the functional, the Dedicated Executives implement platform-dependent low level functions. To coordinate the actions of the user, planner and executive,

an additional component called RobCon (Robot Controller) has been implemented, which functionality extends along the deliberative and reactive layers.

In order to use the architecture with a new robot, the user would need to define a new formal domain and new problems (the inputs of the planner), while the dedicated executives are the only piece of software that is needed to be replaced.

The flexibility offered by ROS in terms of communications and connection of nodes allows to easily modify this architecture depending on the needs of the specific project as illustrated with the two examples provided in this thesis (Figures 5.2 and 6.13).

The first image illustrates an architecture for multiple robots with on-board replanning capabilities.

In this example, ground segments consists of a high level deliberative layer which combine users and the on-ground mission planner, and one instantiation of RobCon.

Ground segment connects with the on-board systems via RobCon, which offers high level functionalities such as uplink plan, start/pause/resume execution, get status, etc.

On-board, each robot has a local replanner and executive connected via a local RobCon, which also helps to synchronize activities between the robots.

Communications between components are organised as follows:

- User ↔ RobCon: The user only interacts with RobCon to request a planning or execution decision. RobCon itself can ask the user to redefine the inputs in case of failure at execution time.
- Planner ↔ RobCon: RobCon can request planning or replanning actions to the planner while the planner only interacts with RobCon to send the results of the planning activity (either a valid plan or failure). In the first case, depending on the level of autonomy, RobCon will proceed to execute the plan at due time (E4) or it will send back the result to the user (<E4), who is in charge of later commanding the execution. In case of failure, the user will be informed.
- RobCon ↔ Generic Executive: RobCon can request the start or stop of the execution while the executive sends back to RobCon an updated version of the plan once the execution is interrupted.
- Generic Executive ↔ Dedicated Executive: Every time a command belonging to a timeline tl is due to be dispatched, SanchoExpress checks in a database to which dedicated executive tl it belongs and sends the command. The dedicated executive will try to execute the commands at due time, reporting back the result of the execution.

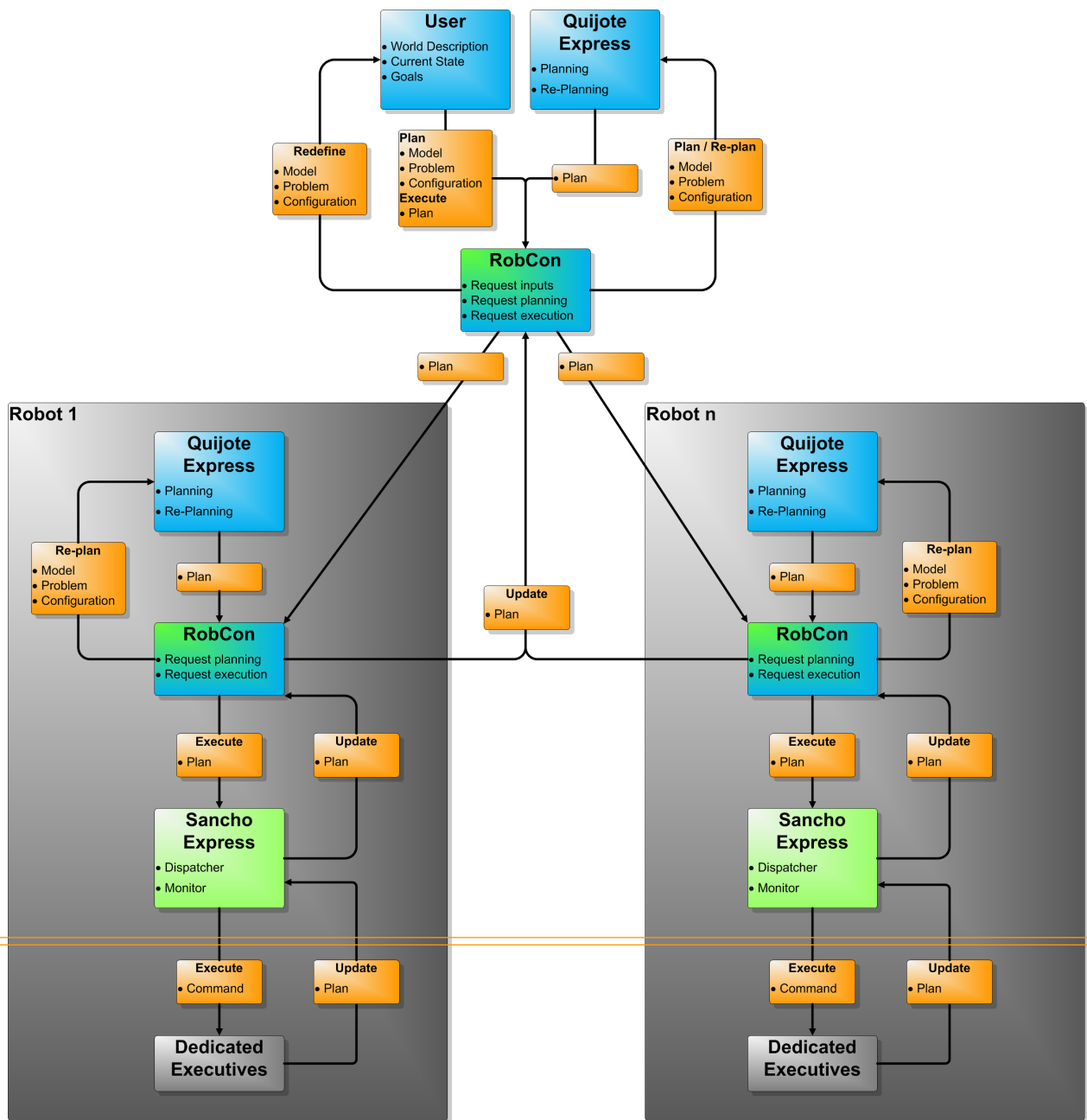
Generally speaking, commands and plans flows downstream, from the deliberative to the functional layer, while observations (updates in the plans and commands) flow upstream.

The life cycle of a plan on-board is presented in Figure 5.3. The first state only indicates a new request to generate a plan. The data (initial state, goals, etc) is stored by RobCon in a queue and waits for the planner. Once it is planned, it is sent for execution. Both planning and execution can be successfully finish, being cancelled, or produce a failure. In this last case, the error will be managed in different ways depending on the level of autonomy as explained in Section 5.2.2.

5.2 Components

Next subsections presents in detail the most relevant components of the QEA architecture.

Deliberative - Reactive



Functional

Figure 5.2.: Multi-rover architecture with replanning capabilities. Each box is a ROS node: Blue boxes represent deliberative components, green reactive and grey functional. The orange boxes on top of the arrows indicate service calls.

5.2.1 User

The higher component in the architecture is the user which is in charge of generating the following inputs: (1) Model of the domain; (2) Problem definition; (3) Configuration of the planner and executive (see Chapter 4). The user has the highest priority to start/stop planning and execution.

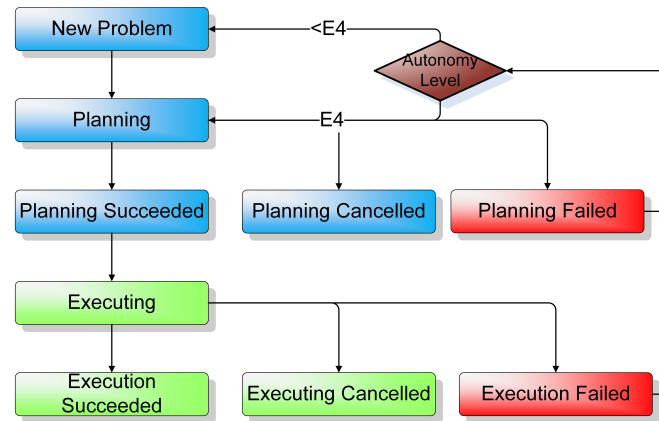


Figure 5.3.: Life cycle of a plan.

5.2.2 RobCon

RobCon (Robot Controller) is the next component in terms of priority. It coordinates the collaboration between user, planner and executive according to the level of autonomy defined on-ground and on-board.

RobCon is implemented as a server running an infinite loop, waiting for planning or execution calls from its clients (users or other RobCon). It is also a client posting requests on the planner and executive. The services offered by RobCon are: *plan_execute(goal)* and *cancel_execution(goal)*.

plan_execute receives as input a goal which format is showed in Table 5.1 and stores it in a FIFO queue, even though it would be easy to add other strategies such as ordering the goals according to their earliest starting time. RobCon peeks the next goal (if any), analyses it taking into consideration the level of autonomy, takes the appropriate action and sends back the result to its client. The goal can specify three different behaviours:

- *command = Plan*: RobCon initializes the planner according to the configuration file and launch it with the domain and problem. Then it waits until the planner finishes with either a failure or a solution, which is returned to its client.
- *command = Execute*: RobCon calls the executive, passing the plan and waits until SE finishes, returning an updated version of the plan with the information resulting from the execution to some reasoning entity (see next paragraph). RobCon will also provide periodical information by means of *feedback* messages coming from the executive.
- *command = Plan_Execute*: RobCon calls first QE. In case it returns a failure, RobCon returns the error to its client. Otherwise, it sends the plan directly to the executive, repeating the behaviour explained before.

In the last two options involving the executive, the behaviour of RobCon in case of a failure during execution depends on the level of autonomy. In E1 level, it can report back directly to its client after the completion of each individual action. In E2 and E3, it executes the plan reporting always the outcome to its client. The difference between the two levels lies on the flexibility inherent to the plan and the capability of the executive to fix some flaws. Finally, in E4 it can communicate directly with the mission planner instead of reporting back to its client to complete high level goals of the plan or repair it after a flaw.

cancel_execution(goal) simply cancels the planning or executing commands.

Variable	Type	Description
domain	string	Path of the domain file.
problem	string	Path of the problem file.
configuration	string	Path of the configuration file.
plan	FlexibleDomainTimeline	Plan to be executed.
starting transition	int	Transition of the plan from which execution should start.
command	<Plan, Execute, Plan&Execute>	Indicates the type of activity requested. For planning, the variables domain, problem and configuration are required, while plan is required for execution.
autonomy	<E1, E2, E4, E5>	Indicates the desired level of autonomy.
feedback status	<Planning, Replanning, Executing, Feedback>	It is possible to ask RobCon about the status of its current command, which can be one of the list.
result	<Planning_Succeeded / Failed / Cancelled; Execution_Succeeded / Failed / Cancelled>	Result after the finalization of the command.
error description	string	If an error occurred, the dedicated executives might attach a description.
plan	FlexibleDomainTimeline	Updated version of the timelines with the status of each action and the time tag in which each was executed.

Table 5.1.: Format of the goal handler parameter used by RobCon.

5.2.3 Planner

The mission planner is in charge of planning and replanning activities. QuijoteExpress, the planner presented in Chapter 4 is incorporated to the architecture as a server waiting in an infinite loop for job requirements coming from RobCon. It offers the following services: *plan(domain, kdb, problem)* and *replan(domain, kdb, problem)*.

The two services are intended to provide the user with different configurations of the mission planner. Planning is typically done on-ground where plenty of resources are available in order to create a new plan from the beginning. In consequence, the planner is configured to produce the best possible solutions using computationally intensive algorithms. On the other hand, replanning use to be performed on-board, where resources are scarce, to fix a plan. The re-planner use to be configured with light algorithms that return the first solution found without exhausting the search space.

5.2.4 SanchoExpress

The executive is one of the cornerstones of any robotic architecture together with the planner. Its mission is to isolate the activities related to the execution of a plan from the details of specific robotic platforms.

Section 2.5 listed some relevant executives. ROS offers some general solutions such as SMACH¹ or Teer², but none of them satisfied the needs of the two case studies. SMACH requires as input a model of all automata defined with its own specific language which would force QE and generally speaking APSI users to duplicate the modelling for the solving an execution layers. Besides, the representation of temporal constraints in APSI and SMACH are not completely compatible. With respect to Teer, it has a closer approach to timelines but it is not consistent with the definition of flexible timeline of APSI. T-Rex has been used with APSI planners in the GOAC experiment [36], but it is more an architecture than an executive which reactor approach is redundant with the nodes defined in ROS. In fact, a version of T-Rex tailored for ROS was promptly discontinued. The Universal Executive might be the most suitable option. It is very expressive thanks to the Plexil[15] language and has been conceived for the space domain. However, it does not directly support timelines. In consequence, a new executive named SanchoExpress (SE) has been developed.

SE is a two-layers MIMO mission-independent timeline-based executive (see Section 2.5) capable of timely **dispatching** actions to the appropriate subsystem and **monitoring** the execution, reporting back to RobCon after completion or failure. It can be used directly by the user in low-levels of autonomy (E1 or teleoperation) or by RobCon in higher levels. SE offers a number of novelties, described below, intended to solve some of the problems that emerge when plans are executed in complex domains. Among them, the most relevant are:

- **Timeline representation:** It supports the execution of Flexible Domain Timelines produced by APSI*. In [139], a flexible timeline is defined as a set of two or more totally ordered time points with associated tuples of values plus a set of minimal and maximal distances between each pair of consecutive time points (see Figure 6.14). The first time point is always the origin (value 0) of the temporal problem while the last is the horizon (H), i.e., if the temporal problem is defined in $[O, H]$, the first time point of the flexible timeline occurs in $[O, O]$ and the last occurs in $[H, H]$.
- **Handle uncertainty:** Derived from the previous one, SE allows to handle uncertainty thanks to the flexible starting and duration times assigned to each action.
- **Designed for ROS:** ROS was chosen as QEA development framework because it resolves a lot of common problems such as synchronization, parallelism, communications, etc. that needed to be explicitly addressed by other approaches. More specifically, SE is implemented as an actionlib client/server, a package that provides tools to create servers that execute long-running goals (such as mission planning or execution) that can be pre-empted. The unit of execution is the action which is divided in three fields: (1) **Goal:** Contains information required by the server to perform the task; (2) **Feedback:** Defines the data provided in case the server is asked to report the progress on the execution of the goal; and (3) **Result:** Sent from the server to the client upon completion of the goal. To the best of our knowledge, SE is the only executive of this type.
- **Two layers:** SE is organized in two layers, both implementing command dispatching and monitoring, but the first one operates at a higher level of abstraction and is domain-independent while the second is dependent.
- **Expressiveness:** The dedicated executives can be implemented with common programming languages such as Python or C++, providing SE with a high level of expressiveness, specially relevant to represent conditional execution and loops. Only Plexil is comparable in this regard to SE.

¹ <http://wiki.ros.org/smach>.

² http://wiki.ros.org/executive_teer.

- Adaptable level of autonomy: The system can behave in different levels from E1 to E4³ just changing the corresponding configuration parameter.

Like RobCon, SE also plays a double role, as server waiting for requests from RobCon in an infinite loop and as a client that makes requests to the Dedicated Executives. It offers the following services: *execute_plan(goal)* and *cancel_plan(goal)*.

Plan execution

execute_plan(goal) is the main function in the reactive layer, as it contains the loop in charge of executing a plan. It receives as input a goal which format is showed in Table 5.2.

Variable	Type	Description
plan	FlexibleDomainTime-line	Plan to be executed.
starting transition	int	Transition of the plan from which the execution should start.
transition Time error description	int	Time at which the last transition has been triggered.
plan	string	Same as before.
plan	FlexibleDomainTime-line	Version of the timelines with the status of each action and the time tag in which each was executed at the time the feedback was requested.
error description	string	If an error occurred, the dedicated executives might attach a description.
plan	FlexibleDomainTime-line	Final version of the plan after the end of the execution (due to nominal ending or errors) with the status of each action and the time tag in which each was executed up to the last transition.

Table 5.2.: Format of the ExecutePlan action used by SanchoExpress.

plan is the most important field of the goal as it contains a translation of the plan generated by QE into a ROS compatible format. The plan is represented as a matrix of values where each row corresponds to a timeline and each column to a transition. For example, the value $values[i][j]$ corresponds to the value of the i – th timeline at j – th transition.

Definition 50 (Transition) *Time point in which at least one of the timelines changes its value.*

When SE receives a new plan, it checks first its internal status. If SE is already executing a plan, the new plan will be queued until it is executed. If the new plan is meant to be executed as soon as possible, then the previous one will have to be cancelled. In case SE is idle, then it performs some internal checks to verify the plan, initializes some internal variables including a dictionary that maps timeline names to their associated dedicated executives and finally calls the Algorithm 7.

Execution starts in the initial transition of the plan. At any given transition t_j , SE extracts for each timeline tl_i the corresponding decision $d_{i,j}$ from the matrix representing the plan and its associated

³ With the exception of E3, as none of the components of QEA support event-tagged commands.

Algorithm 7: se.execute_plan(goal)

begin

```
currentPlan ← goal.plan
currentTrans ← goal.starting_transition
while currentTrans ≤ currentPlan.transitions do
  for  $d_v \in$  currentPlan.values[currentTrans] do
    lastTransition ← getEndTransition( $d_v$ )
     $d_v$ .lowerBound ← currentTrans.lowerBound
     $d_v$ .upperBound ← lastTransition.upperBound
    dedicatedExecutive[ $d_v$ .tlIndex].execute( $d_v$ )
    currentTrans ++
```

starting interval $[lb_j, ub_j]$ which indicates the lower and upper bound to start $d_{i,j}$. Notice that lb_j must be equivalent to the time point of the transition in which $d_{i,j}$ must be executed.

For each of these decisions, the algorithm sets as the execution starting time the earliest value lb_j of the current transition (even though changing it to the latest time would be trivial if required) and as execution ending time the upper bound ub_{j+1} of the value in the next transition, computed by means of the $getEndTransition(d_v)$ function.

Next, SE searches in the local database which dedicated executive is in charge of executing the values of the timeline tl_i and sends a request to execute $d_{i,j}$ at time lb_j . Every time a value is successfully executed, the next transition is checked.

In case the plan is not fully decomposed, the corresponding partially decomposed values $\{d_{sf}\}$ cannot be executed. An additional time range $[lb, ub]$ defined by the user is attached to each of these values, indicating the earliest and latest time before the execution of d_{sf} at which it has to be replanned and therefore fully decomposed. In case no such interval is provided, the executive will rise an error at the time d_{sf} is due for execution and automatic replanning will happen to fix the plan.

The algorithm iterates over all the transitions until either the last transition is successfully finished, it is interrupted by the user or an error arises during execution. Once the execution is completed, SE sends back to RobCon the results.

Example 18 *In the example of the FASTER primary rover, suppose Figure 6.14 represents a solution for a problem in which the rover is initially stopped with the antenna in idle mode and the goals are to do a traverse to certain location and transmit then some data.*

The plan is divided in 10 transitions (T0 to T9) during which at least one timeline changes value. Suppose the Executive is going to process T8. There are two timelines changing value (Locomotion and PathPlanning) and one that does not, Antenna. For Locomotion, the next value to be dispatched is PRLocTraverse, which starting time is equal to the one of T8 and its ending time is the starting time of T9. For PathPlanning, the next value is PRPPIdle which starting time is also T8 while its ending time is not T9's starting time (there is no transition for PathPlanning here) but T9's ending time.

Once SE has extracted the values and execution intervals, it searches in the databases the dedicated executives for their corresponding timelines. In this case, PRLoc is the executive that has to execute the value PRLocTraverse and PRPP the one in charge of PRPPIdle. SE calls both of them in parallel and waits for a reply (either a success or a failure). The process for PRLoc is illustrated in Figure 5.4

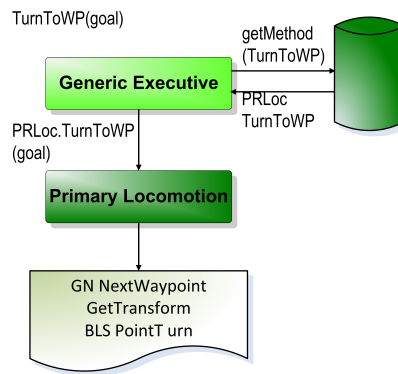


Figure 5.4.: Example of how SanchoExpress executes a locomotion activity.

Canceling a Plan Execution

When invoked, *cancel_plan* stops the execution of the activities being running in every timeline and updates the plan, adding to those activities that were completed a special tag and the time at which they were executed. This information is required to know the initial state of the robot for the next plan. Finally, it returns the control to RobCon which will later on return the updated plan to the appropriate deliberative entity.

5.2.5 Dedicated Executive

SE also includes a template describing how to implement a dedicated executive. It represents an abstract subsystem that offers a list of services mapped to one or several real subsystems. In FASTER five dedicated executives were defined: Communications, Primary Rover Path Planning, Primary Rover Locomotion, Scout Rover Path Planning and Scout Rover Locomotion. Primary Rover Locomotion for example requires services from two real subsystems: Global Navigation and Primary Locomotion System.

Every dedicated executive is implemented as a server running an infinite loop until it is stopped by SE and offers the following services: *execute_goal(goal)* and *cancel_goal(goal)*.

execute_goal(goal) receives as input a goal which format is showed in Table 5.3.

Variable	Type	Description
endTransition value	TimeInstant	Indicates the execution window in which the value must be executed.
	Value	Indicates the value (either a <i>SVValue</i> for StateVariable components or <i>Activity</i> for resources) to be executed.
message	string	Feedback returned by the dedicated executive.
endTransition error description value	TimeInstant	Updated time instant reflecting the time in which the value was actually executed.
	string	Same as before.
	Value	Updated value reflecting its new status.

Table 5.3.: Format of the ExecutePlan action used by SanchoExpress.

Once a goal is received, *execute_goal* searches in an internal dictionary the method in charge of executing the value and calls it. In case the goal is simple, the method just sends the goal to the corresponding sub-system. Otherwise, the method can contain complex structures such as *if – then* conditions, loops and multiple low level operations in order to achieve the complex goal. For example, the method *TurnToWP(goal)*, implemented in the ExecutivePrimaryLocomotion of FASTER to turn the robot to the given waypoint (abbreviated as wp), calls internally the functions *GN_NextWaypoint* provided by the global navigation subsystem, *GetTransform* to change the waypoint coordinates from the world reference to the rover and finally *BLS_PointTurn* provided by Bridget Locomotion System (BLS) to actually perform the turn.

Each low level operation is sent to the controller and the executive waits for its completion. At this level is where the monitoring actually takes place. In case the call fails or the returned values from the call are incorrect, the dedicated executive will label the result as *Execution_Failed*. Finally, the executive returns a message with the updated values to the generic executive.

In the FASTER scenario, some Dedicated Executives were endowed with short-scope repairing capabilities for very specific errors (see Chapter 6). In case one of the methods returns an error for which the dedicated executive has a repairing procedure, it will automatically call it. As the flawed decision d_v has not been replanned, the repairing procedure must be completed within the time allotted to d_v . Moreover, other aspects such as resources consumption must be taken into account when implementing these repair procedures. The time t_e in which the repair procedure finishes will be the one indicated in the timestamp of d_v . In case of succeed, SE will not have even noticed that there were a problem and still it will receive the appropriate values regarding resource and time consumption.

cancel_goal(goal) cancels the execution and returns the value with the status and *endTransition* fields updated.

5.3 Conclusions

This chapter has presented an architecture based on a very flexible and lightweight executive with a completely different approach to previous APSI attempts based on T-Rex.

Focusing on the executive, it is the only timeline executive based on ROS. It is important to remark that T-Rex is not an executive, but rather a system that embeds a planner an executive. Moreover, T-Rex imposes an architecture based on reactors that overlaps with the node philosophy used in ROS, reason why most probably the T-Rex version for ROS did not succeed. Its architecture as a two layers with a generic and dedicated executives is also unique with respect to other alternatives such as Plexil.

The properties used to characterize planners, such as soundness and completeness do not apply to executives as they cannot be guaranteed in real-world applications.

With respect to the three features identified for a planner, performance plays no role for an executive: SE always dispatch as soon as it can the decisions of the next transition and never has experienced delays on this regard.

It is rather the robustness in terms of flexible execution and repair capabilities the most relevant features to take into account. With respect to execution flexibility, SE is based on the most flexible of all timelines that APSI can produce. Regarding its repair capabilities, Chapter 6 will further detail how short-scope plan repair has been implemented for FASTER.

Regarding expressiveness, the possibility to implement dedicated executives with conventional programming languages (C++ and Python) provide SE with an exceptional expressiveness. The possibility to express conditions and loops has positive effects even in the planner, as this complexity can be brought

down to this level instead of the planning modelling language, where such capabilities would be much more difficult to handle.

In the future, a most powerful approach could be implemented, based on a dedicated executive exclusively oriented to repair activities. This executive should be connected to the rest of dedicated executives and could command them directly according to the repair procedure. Another interesting line of research would be to execute directly the Decision Network instead of timelines, helping to improve the collaboration between the deliberative and executive layers.



6 Evaluation in Real-World Scenarios

Rescue activities, maintenance of industrial platforms, surveillance or space exploration are just some of the possible scenarios that could benefit from higher levels of autonomy. QuijoteExpress (QE) and SanchoExpress (SE) have been tested in two of them: a collaborative Mars rover mission named FASTER and a rescue mission, hereafter referred to as USAR. Section 2.2.2 analysed how both scenarios complement each other. Moreover, testing QE and SE with different robots helps to demonstrate the claims in terms of reusability. Due to space limitations, the chapter primarily focus on the first one while the rescue scenario is briefly described.

6.1 Tailoring the Planner and Executive

Thanks to the domain-independent approach of the mission planner and the executive, the adaptation to a new robot or mission is minimal as explained in the following sections.

6.1.1 Adapting QuijoteExpress

The tailoring of QE is reduced to define the inputs of the planner:

- **Domain model:** Formal description in DDL language of the robots and other external components of the environment. The way in which a domain is modelled has a big impact in different aspects of the planner such as performance or plan quality. In temporal planning, and more specifically in APSI, this phase includes the selection of the actions/states that represent the state machines and the definition of the appropriate intervals delimiting the minimum and maximum duration for each activity and transition, which is crucial to deal with the inherent uncertainty of the domain.
- **Behaviours:** Complex behaviours are specific to the mission, such as *TraverseCycle – Team* in FASTER or *Patrol* in USAR. In case the same robot could be used for different purposes, the user would be required to change this input while the domain model could be reused. As an example, in case Hector were used for inspection activities in a power-plant disaster, the behaviour *SearchVictims* could be replaced by *SearchLeaks*.
- **External functions:** Depending on the scenario at hand, the planner might need external functions for different purposes such as estimation of battery consumption, estimation of the time needed for a traverse, etc. In FASTER, an external function will provide QE with an estimation of the number of perception cycles (see Section 6.3.3) and the time required to reach a given waypoint from the initial position.
- **Problem:** Formal description of the present state of the robots and goals of the mission.

The domains and behaviours are based on the concepts of *Component* and *State* described in Section 2.4.4 in which each subsystem is represented as a state machine called Component containing a number of states¹.

¹ The words *state* and *action* will be used indistinctly in this section.

Figures 6.3 and 6.16 illustrate the formal models of FASTER and USAR. Each component is represented as a big soft-blue box containing an automaton, composed by a number of states represented as boxes with unique names. There are five types of states represented in different colours:

- Nominal complex (Blue): State that cannot be directly executed. This state must be decomposed in substates (there might be more than one possible decomposition) by the planner in order to generate an executable plan.
- Nominal primitive (White): State that can be mapped to an action which is directly executable by the subsystem.
- Error (Red): Usually, the execution will be suspended and the replanner will be called to repair the plan. However, under some circumstances (depending on the design) the executive might be able to recover from the error (see Section 6.3.3).
- Default (Orange): Default (components diagram) or initial (behaviours diagram) state.
- Goal (Green): It is only used in the behaviours diagram to indicate the goal state.

Following UML notation, initial states have a triangular shape on the left side and goal states a thick black line around the box.

Regarding relations between states, they are represented as lines, again with different colours depending on the role they play:

- Black: Represent transitions from one state to another.
- Blue: Decomposition of a state in a number of substates and relations between them.
- Dotted orange: Synchronizations (represent dependency) between states of different components.

6.1.2 Adapting SanchoExpress

Similarly to QuijoteExpress, SanchoExpress has been designed as a domain-independent executive. Tailoring it to a specific problem is more complicated than the planner as it implies the development of dedicated executives that interface with the hardware, one for each subsystem (see Section 5.2.5). A dedicated executive has two responsibilities:

- Execution: The executive implements the logic to execute the actions related to each state of its associated subsystem. Some states will be directly mapped to a specific command while others involve a number of them. For example, *PRLoc – Traverse* method contains one single command, *BLS_BridgetFollowPath* used by Bridget Locomotion System (BLS) to move the rover to a given position. On the other hand, *PRLoc – TurnToWP* method calls three commands to perform the action: *GN_NextWaypoint*, *GetTransform* and *BLS_PointTurn*, that is, get the next waypoint, compute the angle between the rover position and the waypoint, and command the BLS to turn the given angle.
- Short-scope plan repair and safe-mode: A fault in the execution of an activity might affect the plan in different ways: Change the sequence of future activities; Recalculate temporal bounds; Recalculate resource consumption; Recalculate parameter constraints; In some situations, a fault does not spread along the plan, and therefore it is possible to fix it immediately without the

need of replanning. To do so, the dedicated executive responsible for the failed action needs to implement the corresponding recovery procedure to resume execution. Otherwise, the executive needs to put the subsystem in safe-mode before replanning is requested. *TraverseCycle – Team* behaviour contains two examples of each scenario. *SRLoc – HazardDetected* (ScoutLocomotion) and *PRPP – ScoutNotFound* (PrimaryPathPlanner) error states are used to recover from the fault and resume execution. As the recovery activities will happen in the time frame assigned to the faulted action, those states with higher risk of failure have been assigned conservative time ranges to support contingency actions if needed. On the other hand, *PRLoc – HazardDetected* (PrimaryLocomotion) and *PRPP – PathNotFound* (PrimaryPathPlanner) imply big impacts in the plan in terms of change of activities, time and resources. In consequence, the corresponding error states will just put the subsystems in safe-mode (in that case, bringing the scout back to the primary), prior to replanning.

6.2 Impact of QuijoteExpress and SanchoExpress Features on the Modelling

Taking advantage of HTLN expressiveness, the domains and behaviours of FASTER and USAR have been modelled in a hierarchy, defining different layers of abstraction (see Figure 6.3 and 6.16). The higher levels contain abstract components with mission-related complex actions while the lower layers have components with low level functionalities directly related to specific subsystems. The main benefits of this approach were introduced in Section 3.3.

In the specific case of FASTER and USAR, it facilitated the distribution of responsibilities among components in a more structured way. As an example from FASTER, the state representing the primary or scout rover reaching a target (*AtWP*) involves the participation of different subsystems and the fusion of data from different sensors. In consequence, it is included as a high level goal in the abstract *PrimaryRover* component. On the other hand, the *Locomotion* component contains a similar state called *Idle* which represents the fact that the robot is stationary at a certain location. However, at this level it is not possible to know whether the robot has reached a waypoint or not.

A significant amount of time was devoted to refine the methods in order to fix or mitigate the impact of uncertainty in corner cases detected during the trials. As an example from FASTER, both rovers were forced to turn towards the next waypoint at the end of each traverse regardless of the instructions of the path planner (which might indicate that this action was not necessary). As it was impossible to ascertain the final position of each rover, this action was intended to avoid some situations where the primary was not able to detect the scout, therefore requiring replanning to bring back the scout to a position visible by the primary. The methods allowed the team to include specific knowledge for the mission that helped to prevent different types of failures in a-priori valid plans.

Similarly, both scenarios benefited from the flexibility of *flexible timelines* to tackle uncertainty. Considerable time was dedicated to define appropriate time ranges (lower and upper bound) for the actions and constraints defined in the domain and behaviours. To set these boundaries, the team ran specific scenarios oriented to measure the time required to perform individual functionalities.

With respect to sufficient planning, this feature was not used on the field trial. Sufficient planning is appropriate when a complex goal can be accomplished in several ways which was not the case of FASTER or USAR.

The executive also contributed to increase plan robustness by means of short-scope plan-repair procedures as explained in Section 6.1.2.

6.3 The FASTER Project

6.3.1 Introduction to Rover Missions

Rover missions (see Section 2.2.2) are considered of maximum scientific interest. Contrary to orbiting spacecraft or stationary landers, they can be directed to interesting features and perform experiments in direct contact with the scientific target.

Space robots and in particular planetary rovers present several challenges, which were analysed in Section 2.2.2. All these factors are reflected in the high number of failed missions to Mars which stands around 50%. Even though most of the problems seem to be solved in modern missions, surface operations still demand a lot of technical advancements, specially from the point of view of autonomy. Among them, Mars Sample Return (MSR), introduced in Section 2.2.2 might be considered as the most important one and, in consequence, a relevant case study for this thesis.

6.3.2 Description of the Mission

FASTER²(Forward Acquisition of Soil and Terrain data for Exploration Rover) is a European Commission funded research project intended to pave the way of future Mars robotic missions such as MSR [171]. Current robotic surface exploration is seriously impeded by the lack of a priori detailed information about the soil and terrain conditions. This is one of the main factors forcing the robots to move slowly, typically below 5 cm/s [16] and, in some cases, to very complicated situations as illustrated by the loss of NASA MER Spirit rover after getting trapped in hidden soft sand. Information about terrain conditions is currently gathered solely from remote sensing instruments such as satellites and rover cameras. However, this data can be misleading due to its low resolution and lack of accurate information about the physical properties of the immediate subsurface layer. FASTER represents an innovative solution to enhance remote sensing data through in situ evaluation of near-surface soil properties conducted by the rover systems. Such data will allow rovers to be safer, faster and more autonomous than it is possible nowadays.

The operations concept focuses on the “traverse phase” required in any exploration missions, more specifically on the long range traverses with minimal science operations foreseen for the sample fetch mission of MSR. It involves two collaborative robots, a primary (Figure 6.1) and a scout (Figure 6.2). The last one is a highly mobile, lightweight (~30kg) rover whose mission is to traverse ahead to assess the trafficability of the preplanned path. If the analysis of the path is positive, then the primary will follow. Otherwise, the scout will explore the surroundings until it finds an alternative route towards the target. To achieve its mission, the scout is endowed with hardware and software subsystems to perform soil sensing. On the other hand, the primary is a heavy (~300kg) ExoMars category rover whose main goal is to perform science. Its payload consists on a combination of soil sensing systems and scientific instruments. During each traverse, it will wait for the scout to find a safe route and then it will follow the path towards the target.

6.3.3 Modelling the Planner Inputs for FASTER

² <https://www.faster-fp7-space.eu/>

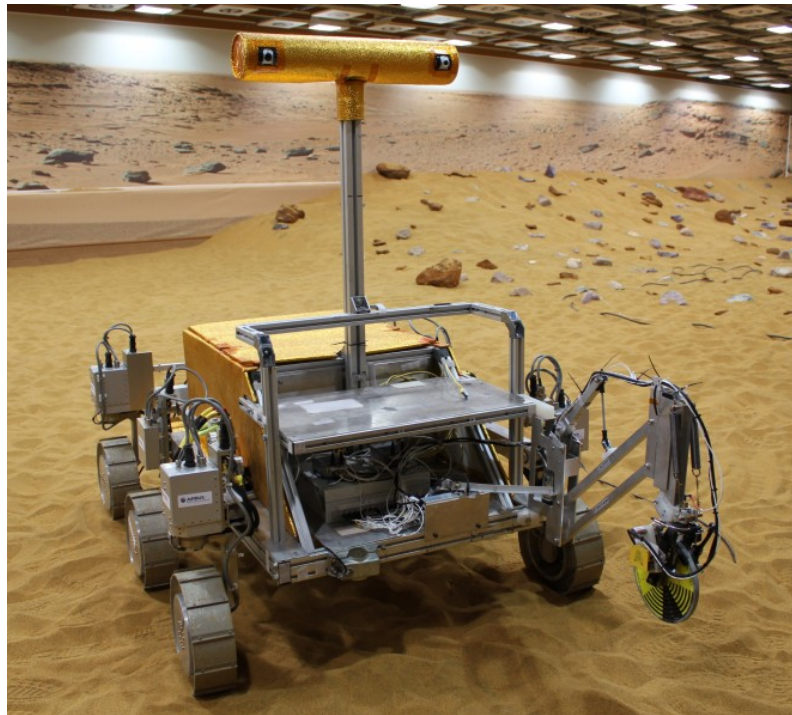


Figure 6.1.: Bridget testbed developed for ExoMars mission. In FASTER, Bridget has been used as the Primary Rover (PR) during the test campaigns in the Mars Yard (Courtesy of Astrium UK).

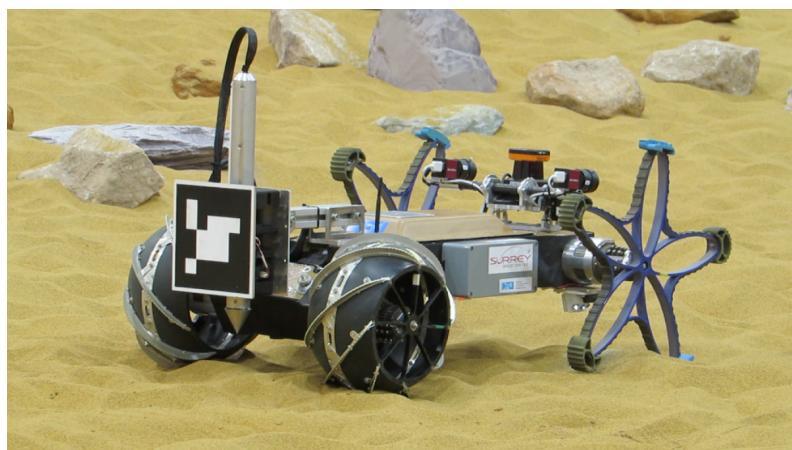


Figure 6.2.: Scout Rover (SR) for FASTER project being tested in the Mars Yard (Courtesy of DFKI).

Domain Model

The domain model contains ten components organised in four groups: complex, primary primitive, scout primitive and external.

The model illustrated in Figure 6.3 is hierarchically organised in two layers. The higher one contains the mission goals to be used by the operators, while the lower one has commands directly related to specific subsystems and is not intended to be directly used by the operator, even though it would be possible if required.

All FASTER components present a start-shape, with the central node being the *Idle* state and the rest of nodes the different services offered by the subsystem. Error states have output transitions but no inputs, that is, there is no path from a nominal to an error state. The reason is that the planner will never require to plan how to achieve an error state. On the other hand, the replanner needs to understand how to recover from a failure, reason why error states have transitions to nominal ones.

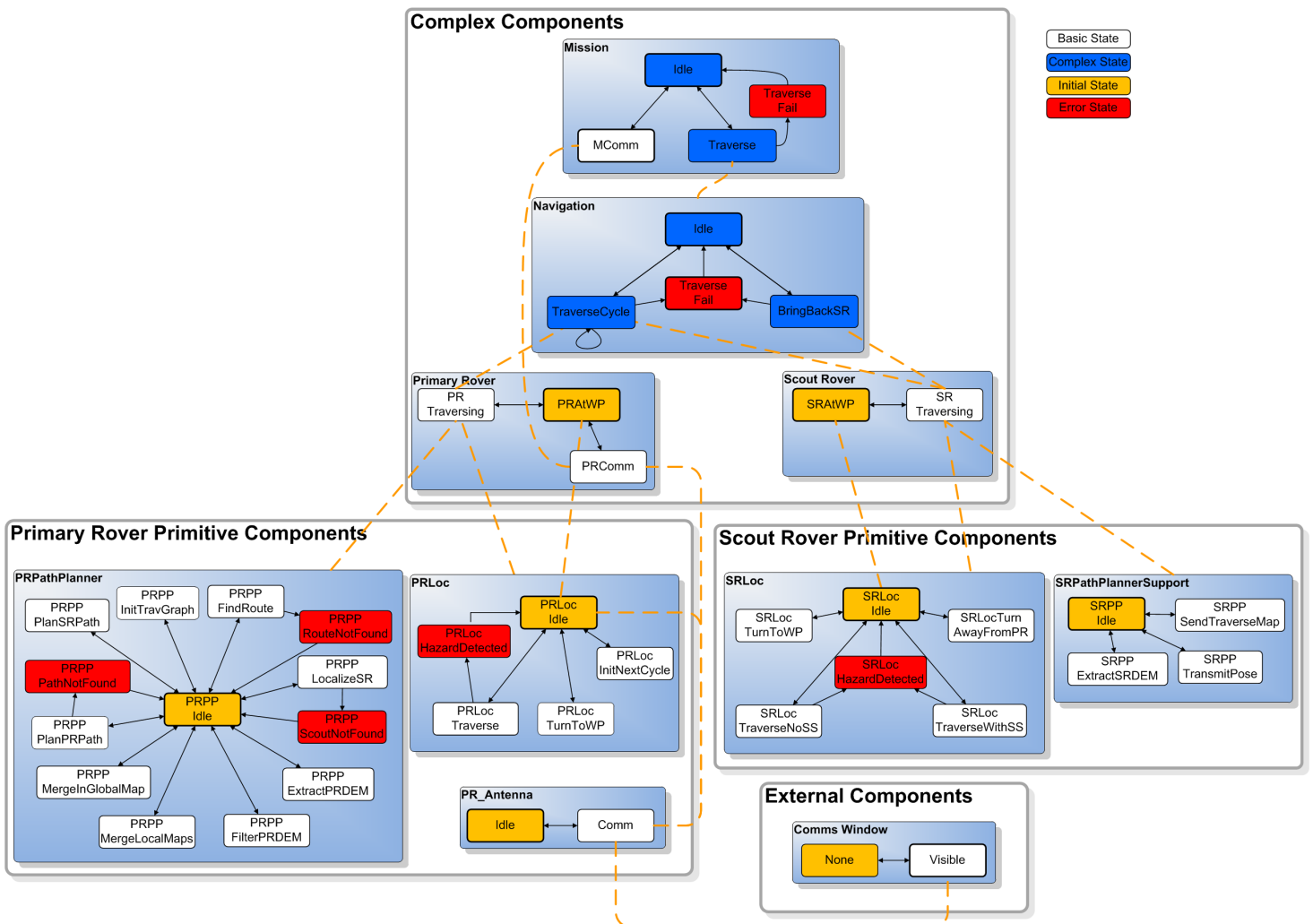


Figure 6.3.: List of subsystems of the Primary and Scout Rover relevant for QuijoteExpress.

For the sake of space, a high level description of the functionalities of each component will be presented without getting into implementation details of the specific states.

Primary Rover Subsystems

The primary rover is responsible for high level deliberation, task execution and supervision. It is divided in five subsystem: mission planner, executive, path planner, locomotion and antenna. The first two do not play any role in the mission plans and therefore do not need to be modelled. The only situation in which the mission planner should consider itself during the generation of a plan could be a situation in which replanning activities could be predicted, in which case the planner could allocate in the plan *replan* activities to take into account the time they require. However, this is not the case for FASTER, where replanning activities only happen as a response to unforeseen events. For similar reasons there are no activities of the executive included in the plan. Again, the only possibility would be to foresee replanning activities, in which case the executive would need to stop execution and later on resume it. The main states of each component are described in the subsequent paragraphs.

PR Mission Planner Subsystem Capabilities:

- **Planning:** Generate a valid plan given a domain description, the present status of each subsystem, a number of goals and optionally a list of behaviours. A plan consists of a number of timelines (one for each component), containing the sequence of actions to be performed by the corresponding subsystem and the time boundaries within which each activity should be executed. Search and heuristic algorithms are computationally intense, as planning for FASTER is performed on-ground.
- **Replanning:** Instead of receiving a list of goals, the replanner obtains as input a failed plan. The goal is to fix the plan introducing as little changes as possible. The algorithms are also different in order to quickly obtain a result rather than exploring all the possibilities. The plan might fail for two reasons: due to a fault during execution or because of the insertion of new goals that are still unsupported in the current plan. This feature (adding new goals) is desirable to allow opportunistic science activities.

PR Executive Subsystem Capabilities: The executive is in charge of two activities.

- **Execution:**
 - **Dispatching:** Upon reception of a new plan, the executive is in charge of dispatching (in parallel) the activities of its associated timelines preserving the temporal constraints.
 - **Cancel execution:** The user or the Monitor might request to cancel the execution. The executive should then stop each subsystem in a safe mode.
 - **Recovery:** Some errors which do not represent changes in the temporal constraints can be fixed by the executive.
- **Monitoring:** In charge of fault detection (not recovery). In case an activity fails to execute, cannot start or finishes at the appropriate time, the Monitor will report an error.

PR Path Planner Subsystem Capabilities:

- **Traverse Graph operations:** Graph search and maintenance operations allowing the addition/deletion/modification of vertices and edges in the traverse graph.
- **Mapping:** Build Digital Elevation Maps (DEMs) obtained from merging data from multiple stereo images, including those from the scout rover. Additionally, images and/or data from the primary rover should be filtered to ensure that the scout rover is not included as part of the elevation map.

-
- Path planning: Planning a path to a local goal based on a Digital Elevation Map (DEM). Apart from geometric obstacles and rover capabilities, this should take into account the rocks detected by the visual soil sensing component of the FASTER Soil Sensing System (SSS) and any known trafficability information.
 - Self-localization: Accurate self-localization is required for successful, long range traverses of pre-planned paths and to identify if the target location has been reached.
 - Scout localization: Recognition and localization of the scout rover using camera images from the primary rover.

PR Locomotion Subsystem Capabilities:

- Path traverse: Following a planned path safely, while interacting with components of the FASTER SSS that have been deployed on the primary rover.
- Point turn: Allows the primary to orientate itself with no requirements in terms of trafficability, as the rover will not change position respect the x and y coordinates.

PR Comms Subsystem Capabilities:

- Communication: (1) Transmission of commands to the scout rover and reception of trafficability information and other data (such as stereo images and status); (2) Transmission of telemetry to a satellite and reception of new plans.

Scout Rover Subsystems

The scout rover has very limited responsibilities in terms of high level deliberation. It has three subsystems: executive, path planner and locomotion. The executive is considered as a black box that communicates only with the executive of the PR translating the telemetry and commands to TCP packages. Its functionalities are similar to the dispatcher of the primary (without monitoring).

SR Path Planner Subsystem Capabilities:

- Self-localization: While it is expected that the primary rover will provide periodic localization updates to the scout, the SR is also capable of self-localization.

SR Locomotion Subsystem Capabilities:

- Path traverse: Moves the scout along a path while conducting “forward sensing” operations with the components of the FASTER SSS on-board the scout rover.
- Return to primary rover: In certain cases such as hazard detection during traverse or some emergency scenarios (identified in [AD2]), the scout rover should be able to return to the primary rover using the local DEM and the last path followed (or an updated path received from the primary rover).

Abstract subsystems

Abstract components are used by operators to define mission goals. FASTER has four abstract components: Mission, Navigation, Primary Rover and Scout Rover. The Mission component contains all high level goals related to FASTER mission. In nominal situations, the user should define goals exclusively for this component and the planner will take care of adding the corresponding actions to achieve them. In non-nominal situations such as failures, the operator might want to specify lower level goals to have a

better control of the plan generated. Due to the relevance of traverses for this scenario, a dedicated component called Navigation has been defined. With respect to the rovers, each has an abstract component with high level goals.

Mission:

- *Traverse*: Involves all activities related with the preparation of the traverse (localization, map generation, manoeuvres calculations) and locomotion activities for both robots from the starting waypoint until the destination is reached or a failure is raised.
- *Communicate*: Involves the transmission/reception of information between the primary and scout or the primary and a satellite.

Navigation:

- *TraverseCycle*: This complex activity might be achieved in different ways depending on the scout availability. It involves a number of subactivities that are repeated in a cycle until the final waypoint is reached. As indicated in Chapter 4, representing loops is a novel feature critical for FASTER. Section 6.3.3 explains how it is used.
- *BringBack – SR*: In case the scout finds a hazard that cannot be avoided, it should come back to a position close to the primary.

Primary: The state of the primary can be: *Idle*, *Traversing* or *Communicating*, each synchronized with a number of subsystems.

Scout: The scout has two states: *Idle* and *Traversing*.

External subsystems

Besides the robot description, the formal model must contain as well relevant information about those external elements that might pose constraints to the mission. The surface (in this case of Mars) is not included, as it is directly managed by the Path Planner subsystem. Typical external components are ground segment, satellites, other robots, etc.

Satellite Communications Subsystem:

- Communication windows: Communications (up and down) using the primary antenna are bound to the windows in which the satellite is visible. In consequence, the satellite is modelled as a component with two states: visible or not. The communication windows are known in advance, therefore they can be specified in the problem as a list of facts including the starting and ending time in which the satellite will be visible.

Behaviours (Knowledge DataBase)

Besides the domain model, a number of hierarchical behaviours have been defined. While the domain is used to formally describe the world in terms of components and states, the behaviours are used to describe mission related concepts, more specifically how to combine low-level actions to achieve complex goals. The behaviours presented below have been implemented in DDL and passed as an input to QE.

Two high level behaviours have been considered for FASTER: *Traverse* and *Communicate*, each describing how to achieve their corresponding complex goals.

Communicate is simulated via wireless connection between the primary rover and a laptop that plays the role of a satellite, with the communications being limited to time windows that simulate the pass of

the satellite over the area in Mars where the rover is. This activity is relevant for the planner because it imposes realistic constraints on the mission. Besides, it helps to test the planner in a very easy manner under different stress levels.

Traverse is more complicated and the main target of FASTER. Its behaviours are organized in a tree-structure where higher levels involve complex behaviours that are decomposed in more detailed subbehaviours as represented in Figure 6.4.

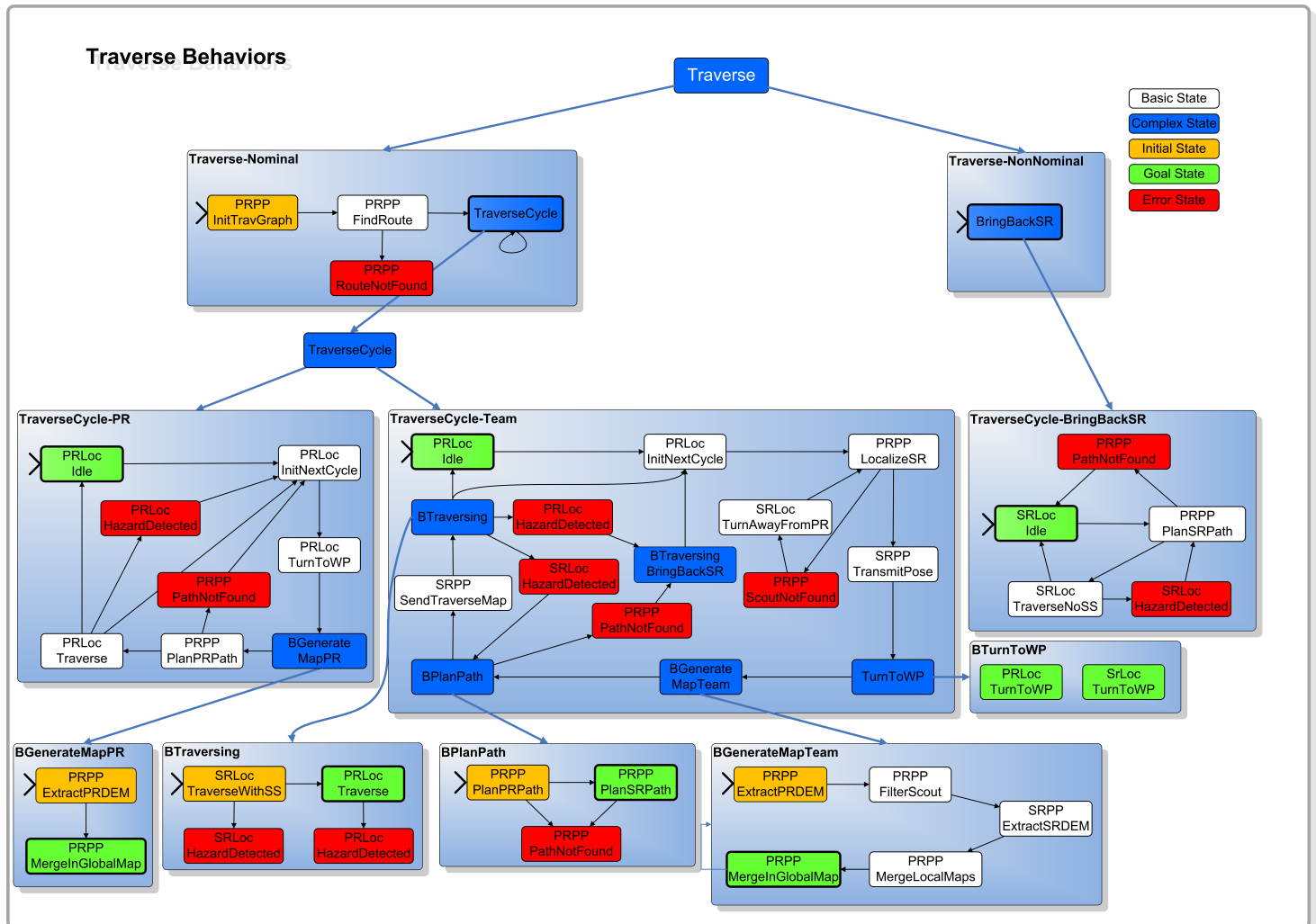


Figure 6.4.: List of hierarchical behaviours included in the KDB of QuijoteExpress for FASTER.

Assumptions and Requirements

Suppose a nominal scenario in which the two robots are in formation with all their subsystems in the default state (Figure 6.5).

Data of the Martian surface is used by Ground Control to identify potential paths (a preferred path as well as several other “contingent” paths). Each planned path is represented as an nondirectional graph where the nodes are waypoints and the edges potential straight line paths between waypoints (Figure 6.6). Each edge has an associated cost that can be estimated based on distances, even though actual costs are influenced by other factors (such as the slope of terrain, expected terrain traction, etc.). Calculation of the best (global) path is then a graph search problem performed at the beginning of each traverse and after a graph update.

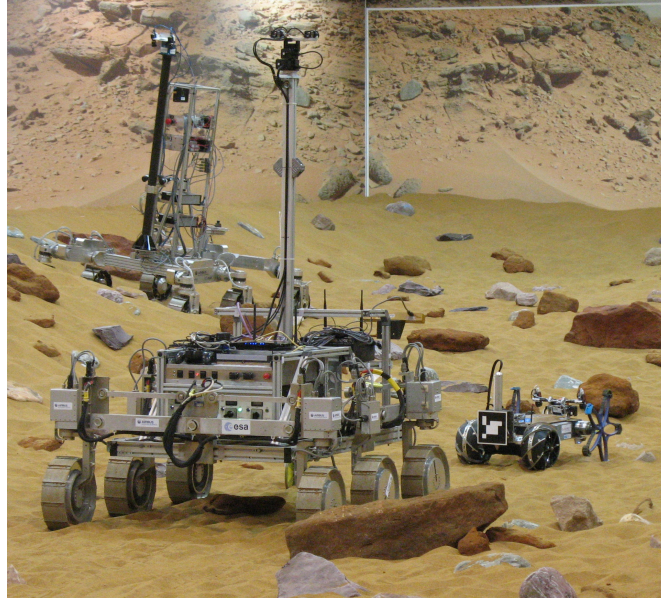


Figure 6.5.: Primary and Scout Rovers in formation before starting the next traverse to waypoint (Courtesy of Airbus).

Traverse between waypoints

A traverse can be initiated in case the previous step has been successfully achieved. Figure 6.4 illustrates three different behaviours included in the KDB for auto navigation with the primary and scout rovers:

- *TraverseCycle – PR*: Describes the activities to conduct when the primary is traversing alone, used in case the scout is lost.
- *TraverseCycle – Team*: Describes the activities to conduct when both primary and scout are navigating in nominal formation.
- *BringBack – SR*: Describes the activities to bring back the scout to a position close to the primary. This behaviour is required in situations in which the scout did not find a path, the primary cannot traverse the planned path towards the scout or there is a change in the mission goals when the two robots are not close.

Change from team to primary-alone traverse must be manually commanded, as the lost of the scout represents a major failure that should be assessed on-ground.

Focusing on *TraverseCycle – Team*, it moves the robots from one waypoint to the next until the target is reached. Two waypoints might be tens or hundreds of meters apart. However, rovers sensors have a limited range of approximately 4 meters. In consequence, a traverse between two consecutive waypoints is split in several parts using intermediate waypoints that are separated at most 4 meters away. Each individual traverse to reach an intermediate or final waypoint is achieved by a sequence of actions named *PerceptionCycle* which is depicted in the external cycle of the *TraverseCycle – Team* behaviour (see Figure 6.4), from *PRLoc – InitNextCycle* until *BTraversing*:

1. *PRPP – InitTraverse*: Initializes the global navigation traverse graph with the new waypoints and goals provided from Ground Control.

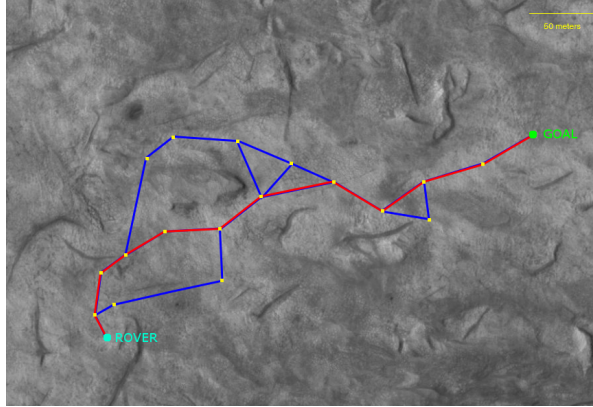


Figure 6.6.: A set of possible paths across a sub section of the Mawrth Vallis, which was selected as a possible landing site for the MSL. A preferred path is highlighted in red (Courtesy of Airbus).

2. *PRPP – FindRoute*: Selects one of the possible routes passing by all waypoints from the traverse graph.
3. *PRLoc – InitNextCycle*: Transition from the previous perception cycle to the next one.
4. *PRPP – LocateSR*: At the beginning of each cycle, both rovers should be relatively aligned and separated by a safe distance. In this situation, the PR Navcam (mounted in the chassis) acquires stereo pictures that are processed to identify the scout marker (the black box in the rear part of the SR in Figure 6.5) and then to determine the scout position.
 - *PRPP – ScoutNotFound*: In case the scout is not properly aligned or it is farther than expected, the primary might not be able to localize it. In this situation, the dedicated PrimaryPathPlanner executive can automatically fix the problem without requiring replanning. The scout is commanded to position itself in front of the primary by means of the *TurnAwayFromPR* action and *LocateSR* is repeated. These activities take place in the time allocated for the initial *LocalizeSR*, which is given a conservative time to prevent replanning in such situations.
5. *SRPP – TransmitPose*: The PR sends to SR its estimated position.
6. *PRLoc – TurnToWP*, *SRLoc – TurnToWP*: Starting from a waypoint, the rovers turn in place until they are facing the next waypoint. To do the turn, the rover first retrieves the next waypoint from the route computed in *InitNextCycle*. Next it gets the rover current position and calculates the desired angle. Finally, the low level locomotion function is called to perform the turn.
7. *BGenerateMapTeam*: Complex state decomposed in the following subactivities:
 - a) *PRPP – ExtractPRDEM*: The stereo cameras take three pictures (left, front and right of the rover) that covers an area of more than 90 degrees. The result is a point cloud from which a DEM is generated.
 - b) *PRPP – FilterScout*: 3D information in the DEM that corresponds to the scout rover is removed, so that the scout is not detected as an obstacle.
 - c) *SRPP – ExtractSRDEM*: The SR extracts a point cloud of the terrain in front moving the laser up and down.

-
- d) *PRPP – MergeLocalMaps*: Merges filtered point clouds from the PR and SR generating an extended DEM which is also more accurate in those areas where the two point clouds overlap (Figure 6.7).
- e) *PRPP – MergeInGlobalMap*: In case initial information of the area around the rover is provided in advance, the global map is represented as a low resolution map; otherwise, it is empty. As high resolution local maps are generated, they will replace the corresponding areas of the global map (Figure 6.8).
8. *BPlanPath*: The path planner gets information from two sources: the trafficability, provided by the scout (see next point), only available once the scout has finished its first traverse and the DEM. The cost values from each map are weighted and fused into a single value which is then used by the path planner, based on an A* algorithm to compute the most cost-effective path from the current position towards the next waypoint (see Figure 6.9).
- *PRPP – PathNotFound*: The PrimaryPathPlanner cannot find a path towards the next waypoint. In that case, the scout is commanded to come back if it is not close to the primary. Once together, the traversability graph needs to be updated. Depending on the position of the rovers, there are two possibilities: (1) Rovers are located in a waypoint: The edge representing the path between the last (or current) waypoint and the next way point is removed. The global path can be considered invalid and an alternative path to the final destination needs to be selected; (2) One of the rovers is not at a waypoint: The current location of the rover is inserted into the graph as a new vertex. Knowing that the robot has reached the new position from the last vertex, it is safe to add a new edge between that last waypoint and the newly created one. The edge for the non-traversable path is removed. Again, a new graph search should be run to determine a new path.
- Finally, to avoid replanning, a dedicated executive (PrimaryPathPlanner in this case) calls the generic executive to drop the remaining activities of this perception cycle and start the next one. The path planner should also overestimate the number of perception cycles required to reach the target (higher than expected) to avoid replanning (see Section 6.1.1). At the time of FASTER final demonstration, the executive was not yet endowed with the capability of dropping activities and therefore this feature was not used.
9. *SRPP – SendTraverseMap*: The updated DEM including the path is sent to the scout.
10. *BTraversing*: In case the path planner cannot find a path for either the PR or SR, it calls the scout back and repeats the cycle of operations from the beginning. Otherwise, it commands the scout to start moving towards the waypoint. During the scout traverse, the laser scanner is used to detect obstacles. At the same time, trafficability information is gathered from the WLSIO (Wheel-Leg Soil Interaction Observation) system and the mDCP (motorized Dynamic Cone Penetrometer). WLSIO is a Soil Sensor System of the SR that measures the sinkage of the front wheels with two cameras and positions this measurement in the local map using information from two IMUs and wheel odometry. If the trafficability values go under a certain level, the scout stops and sends the information to the primary. A new path is then computed to take into account the new information. If the trafficability value is within a range called *maybe*, the rover deploys the penetrometer or mDCP, a tube ended in a cone that can be actuated to impact the surface and measure the sinkage. The results of the mDCP are used to produce the final Go/No-go value. When the scout arrives at the waypoint, the primary follows. In case it encounters an obstacle that cannot be avoided, then the scout is brought back and a new cycle starts.

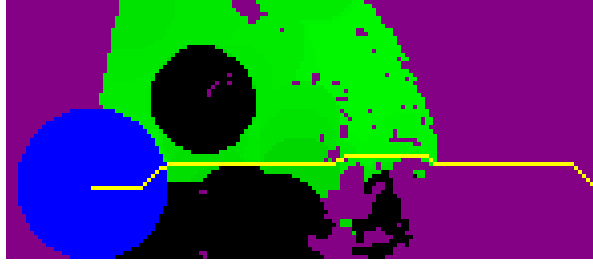


Figure 6.9.: Initial path for the primary rover (Courtesy of SAS).

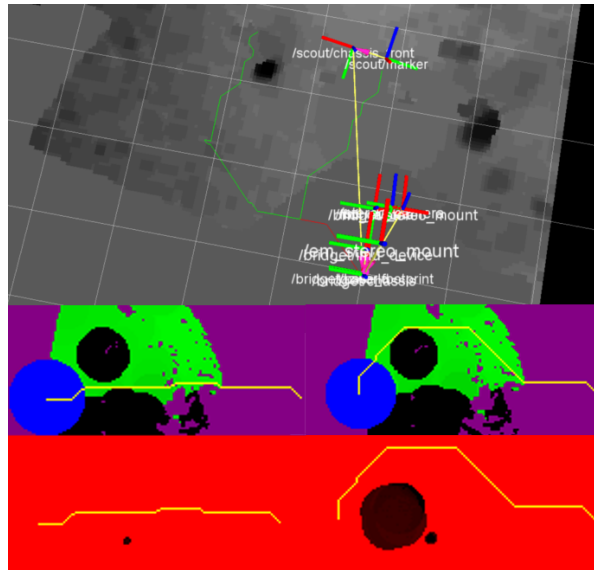


Figure 6.10.: Replanned path for PR and SR after successful traverse of the SR (Courtesy of SAS).

Ground Control. In a MSR scenario, if communications cannot be established, a contingency plan could be adopted where the cache (or MAV) location is taken as the goal waypoint.

- The rover reaches the goal waypoint, which indicates a successful execution of the traverse plan. In case this is not the last activity, execution continues. Otherwise, the rover enters Idle mode until further instructions arrive.

External Functions

When the mission planner constructs a new traverse plan, it requires from the path planner some information, namely estimated traverse time and number of perception cycles. However, this information is still unknown, as the path planner also needs some information produced by activities not yet planned in order to compute a path. Two external functions were used to provide estimations for these two parameters.

The first one is a conservative estimation using as distance the straight line between the waypoints, the average speed of the rover ($3\text{cm}/\text{sec}$) as speed, and a certain penalization k to take into account point turns (which consume a lot of time) and possible deviations to avoid obstacles. The resulting equation $t = \frac{\sqrt{(x_{dest}-x_{orig})^2+(y_{dest}-y_{orig})^2}}{v=3} \times k$ gives an estimated time that will be used to define the time bounds of each perception cycle.

The second divides the distance between two consecutive waypoints by the maximum range of the sensors (4 meters), that is: $num_{cycles} = \frac{\sqrt{(x_{dest}-x_{orig})^2+(y_{dest}-y_{orig})^2}}{range=4}$. This number will be used by the mission planner to allocate num_{cycles} times the activities related to the perception cycle.

Problem

A problem must define the initial status for each subsystem (*facts*) and the desired status to achieve (*goals*). In FASTER, goals are locations that represent waypoints where the rovers should go. Problems are implemented in PDL language and represent one of the inputs of the planner together with the domain, behaviours and external functions.

6.3.4 Implementing the Dedicated Executives for FASTER

Each of the components in the lower layer of the model presented before is directly related to a rover subsystem. Only four of the ten components in the model have a dedicated executive: PR Path Planner, PR Locomotion, PR Antenna, SR Path Planner and SR Locomotion.

SR Locomotion is specially relevant as it is the only one containing short-scope repair activities, triggered in case the error state *SRLoc - HazardDetected* is reached. As indicated in Section 6.1.2, the error states in PR Locomotion and PR Path Planner are too complex to be fixed by the executive and require the mission planner.

6.3.5 Evaluation

Testing

In order to achieve the goals of the project, validation & verification became crucial at each stage of the development. As the components and subsystems were integrated, the emphasis evolved from the initial hardware testing to the software integration.

Subsystem-level Testing

Each subsystem was individually tested by means of **unit tests** in order to validate all the requirements prior to the integration phase. Once each of them had acquired the necessary maturity, the integration between them became the focus. The integration of the FASTER (hardware) sub-systems occurred over the course of 5 campaigns between April 2014 and August 2014, each lasting an average of 4 to 5 days.

Integration of the software modules was still not feasible at this stage. In the case of the planner and executive, several unit-tests were defined in order to verify the model, behaviours, plans and executability. Validation and verification was specially complicated because the tailoring of both components (see Sections 6.1.1 and 6.1.2) has strong dependencies with respect to the subsystems.

In the case of QE, the model and behaviours continued evolving until the formal definition of each subsystem was completed. Both were manually verified first with the FASTER team and later on checked against the DDL parser. Different problems were defined for the three main traverse types (Team, Primary, BringBack-SR) and the plans obtained were remotely evaluated against a simulator at SAS (Belgium). The tests conducted for QE can be classified into two groups:

- Fully defined plan: For the Primary and Team traverses, a plan like the one showed in Section 6.3.3 was used. Simulations allowed to improve the temporal bounds assigned to activities, specially relevant for those that could lead to error states, and more importantly to early detect conceptual errors in the behaviours. No partial solution was required in this scenario, as the rovers did not have on-board replanning capabilities.
- Replanning: For those cases in which execution failed, the capability of the planner to fix the plan was verified. The activity consisted in modifying the problem to reflect the new status of the robots, obtain a new plan and uplink it. Like in the previous point, simulations helped to improve the model and behaviours.

Figure 6.14 shows the timelines extracted from the plan for the *TraverseCycle – PR* behaviour. The plan obtained for *TraverseCycle – Team* will be analysed in Section 6.3.5.

V&V of the dedicated executives was more complicated, because it required a high level of detail. During initial phases, dummy methods were used to simulate the behaviour of the real systems in order to test the executive in the simulator at very early stages of the different subsystems. The tests conducted with the simulator were:

- Nominal mission: Execution of a plan goes as expected.
- Nominal mission with hazards detection during traverse: Replanning is not required unless the estimation of perception cycles is incorrect.
- Non-nominal mission: Execution fails due to a failed activity (*PRPP – ScoutNotFound* or *SRLoc – HazardDetected*). Replanning not required, but executive needs to conduct repair activities.
- Non-nominal mission: Execution fails due to a failed activity (*PRLoc – HazardDetected* or *PRPP – PathNotFound*). Replanning is required due to the big impact of repair activities on the plan.
- Non-nominal mission: Execution fails because the maximum time allowed for a goal was exceeded. Replanning is required.
- Non-nominal mission: Execution fails because extra activities have to be allocated in the plan (more perception cycles). Replanning is required.

Mission-level Testing

Once the various subsystems were integrated and the software interfaces provided, the higher level functions at a mission level could be tested in the Airbus Defence & Space Mars Yard, also used during the final demonstration. This is a 13m x 30m indoor facility that represents a mock-up of a Martian-like environment that provides consistency and repeatability to test robotic prototypes (Figure 6.11).

Four test campaigns between July and October 2014 were dedicated to specific integration and development activities comprising: Handshake between the various systems; Functional verification of the key functions; Full system testing against a representative mission scenario; Final demonstration.

Generally, each test consisted of three steps:

1. Rovers placement and on-board flight software initialization (navigation, path planning, etc).
2. Prepare the test area (e.g. preparing soil density, situating hazards, etc).



Figure 6.11.: Panoramic picture of the ASU Mars Yard with two Bridget rovers and the control room in the far side (Courtesy of Airbus).

3. Launch the test, gathering all relevant information for further analysis.

From the point of view of QE and SE, the tests conducted were those already validated in the simulator. The only changes required came from the replacement of the dummy services by the real ones in the dedicated executives and the fine-tuning of the time bounds.

Final Demonstration

The final system was presented to EU, ESA and NASA officials during the full day Workshop and Final Demonstration event, held on October 23rd, 2014 in the New Mars Yard facility, Airbus DS, Stevenage, UK.

The demonstration was based on the following scenario:

- The mission targeted a location 15m away in straight line from the starting point.
- The path to the target contained both visible, hard obstacles (e.g. rocks) of different sizes and hidden hazards considered unsafe for the PR locomotion system.
- The rocks were placed to force the PR to go through the hidden hazard zone, comprising a sub-surface hazard consisting of small pit(s) containing a Sand Trap Analogue covered with a layer of sand to make it visually similar to its surroundings.
- The planned path should be first traversed by the SR, whilst deploying its sensors. The scout should detect the hidden hazard, evaluate its severity using its on-board sensors, and convey data to the PR with a trafficability evaluation and the location of the hazard.
- Data on visible hazards would be derived both from the SR on-board sensors and from PR data

The demonstration was specially relevant for this thesis for two reasons: It used QE to generate plans for a real system in a real environment; SE demonstrated the execution of a plan and repair capabilities.

Preparing the Test

Setup of the Mars Yard: The configuration of the scenario, described in detail in [175], is shown in Figure 6.12. The departure point was placed close to the control room, while the destination point was placed in the other side of the Mars Yard, some 15 meters apart in straight line. A sand bank and a trap, not visible with cameras, were situated in the way between the rovers and the target. Other small rocks were placed according to the normal distribution found in Mars. Finally, a big rock represented the second obstacle after the trap in the way of the rovers. The potential routes that the operators estimated

beforehand for the mission are displayed in different colours where red lines represented non-trafficable paths, yellow lines tentative paths and green are the estimated final paths.

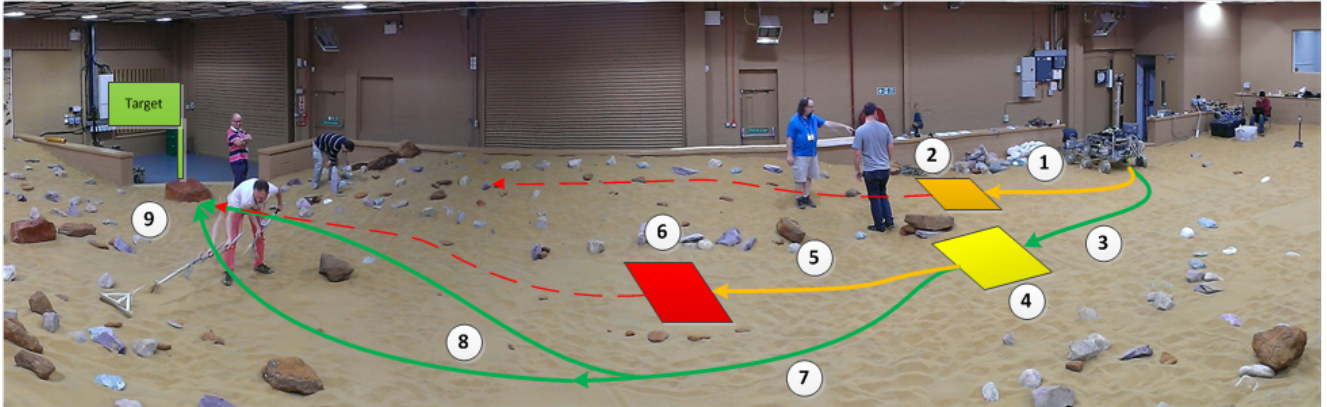


Figure 6.12.: Representative Mission Scenario to exercise all the FASTER functions (Courtesy of Airbus).

Setup of the Rovers and Computers: The two robots were initially in formation, with the scout in front of the primary pointing toward the target. Each of the three operators were connected to the On-Board Computer (OBC) to initialize the different subsystems and launch the plan. Once the demonstration had started, connection was maintained to abort execution in case of critical failures.

Setup of Ground Control: The mission planner was deployed in an external laptop, while the rest of the software for each subsystem (including the executive) was running in the primary OBC. Field trials required the participation of at least one responsible for each subsystem: PR On-Board Computer (OBC), SR OBC, Planner/Executive, Path planner, Wheel Bevameter, Penetrometer, Primary Rover, Scout Rover, etc. Upon starting of the tests, only three operators were required: Primary OBC, Scout OBC and Planner/Executive. Once the two OBCs were ready and a plan generated, the Executive operator could launch the plan, starting the autonomous operations.

Setup of the Mission Planner and Executive: Figure 6.13 illustrates the 3-tier architecture deployed for FASTER. The deliberative component, a mission planner located in the Ground Segment represented by a laptop, was in charge of generating the initial plan. The reactive component, a generic executive installed on the the on-board computer (OBC) of the PR, was in charge of commanding both rovers. The approach derives from the idea of a master robot in charge of execution and FDIR functions according to the uploaded plan (taking advantage of higher computational capabilities) and a subordinated robot following instructions. This schema is very simple and allowed the mission to reach the targeted E3 level of autonomy, similar to current MER or Curiosity missions. Nevertheless, the level of autonomy could have been easily increased to E4 just by deploying a replanner on-board and reconfiguring RobCon as defined in Figure 5.2.

Problem Definition: The mission consisted of just two goals: a long traverse and a final communication with ground segment, both marked to be fully decomposed on planning time. As indicated in the formal specification presented below, all subsystems are set to the default state. Notice that f_9 and f_{10} define the communication windows in which the satellite is visible. The first goal (Navigate) has two parameters. $?f_iter$ is an external function used to estimate how many perception cycles are required. $?t_traverse$ defines the traverse mode (Team, Primary or BringBack-SR). Finally, the second goal defines a communication that should happen after the traverse.

```
PROBLEM Rover-Problem (DOMAIN FASTER_Domain) {
//-----
```

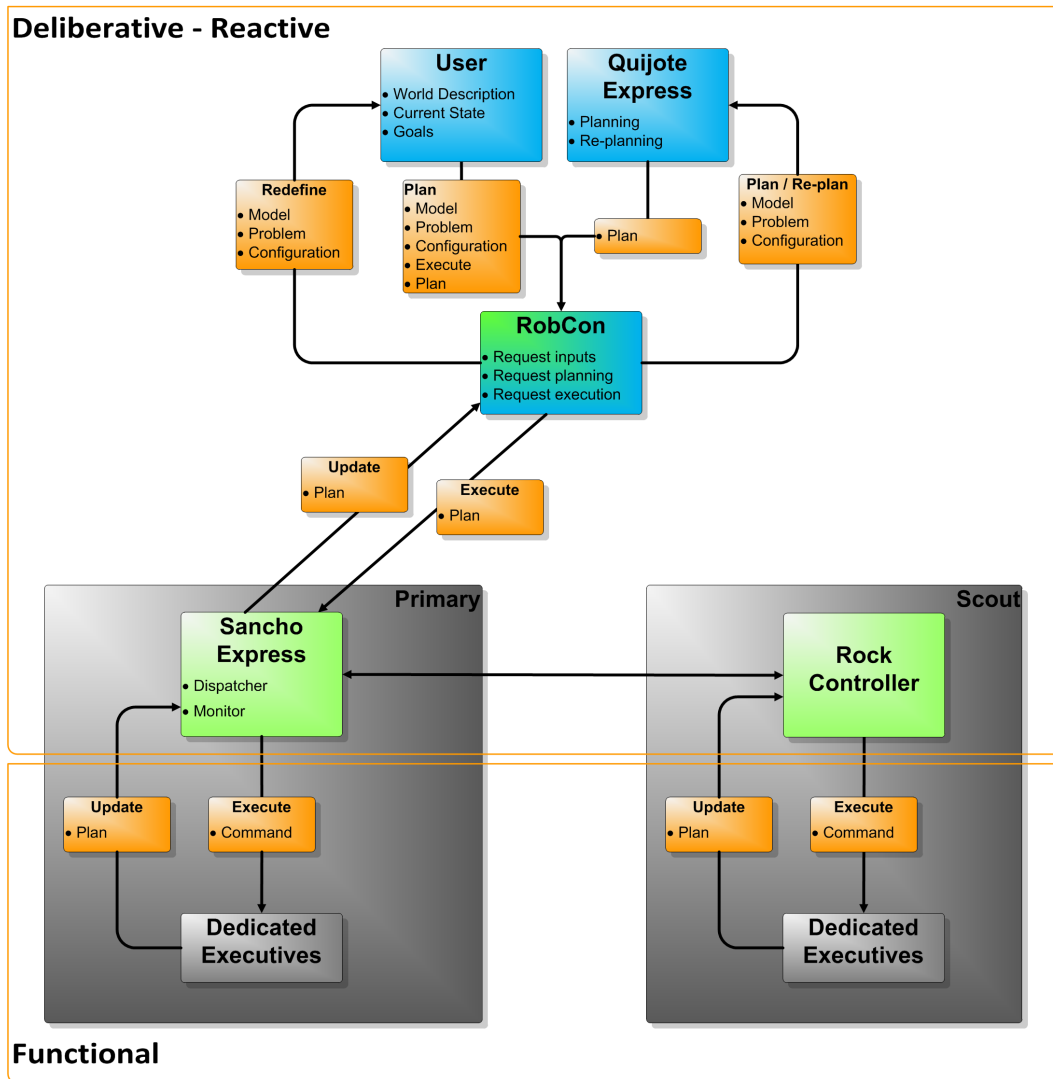


Figure 6.13.: Architecture for FASTER scenario with two collaborative rovers.

```
// Facts
//-----
f1 <fact> compMission.t11.Idle();
f2 <fact> compNav.t11.Idle();
f3 <fact> compPrimaryPath.t11.Idle();
f4 <fact> compPrimaryLoc.t11.Idle();
f5 <fact> compPrimaryAntenna.t11.Idle();
f6 <fact> compScoutPath.t11.Idle();
f7 <fact> compScoutLoc.t11.Idle();
f8 <fact> STATIC compCommWindow.t11.Visible();
f9 <fact> STATIC compCommWindow.t11.Visible();

f1 RELEASE [0, 0];
f2 RELEASE [0, 0];
f3 RELEASE [0, 0];
f4 RELEASE [0, 0];
f5 RELEASE [0, 0];
f6 RELEASE [0, 0];
f7 RELEASE [0, 0];
f8 AT [0, 0] [600000, 600000] [600000, 600000];
f9 AT [3000000, 3000000] [3600000, 3600000] [600000, 600000];

//-----
```



```

// Goals
//-----
g1 <goal> compMission.tl1.Navigate(?f_iters, ?_inTeam = TRUE);
g2 <goal> compMission.tl1.Communicate(?gfile2 = 1);
g1 RELEASE [1, +INF];
g2 RELEASE [2, +INF];
g1 BEFORE g2;
}

```

Results

QuijoteExpress was configured in single-thread mode (no parallelism), to search for all possible solutions, to use the resolvers defined in Section 4.3.3 (forward-chaining unfold, decomposer and scheduler) and to use blind heuristics. The type of output was a flexible plan in which starting and duration times were described as intervals.

Three inputs were defined: the domain shown in Figure 6.3, the navigation behaviours shown in Figure 6.4 and the problem previously described.

The domain was modelled taking advantage of the hierarchical approach of HTLN, facilitating the definition of abstract components such as the Mission planner, which states became the only goals defined for the trial. In addition, the HTN capabilities were also exploited to model the mission behaviours as HTNL methods. With respect to the problem defined for the trial, it was not very challenging from the point of view of planning. The deterministic nature of the decompositions and the role of QE as on-ground planner made unnecessary to use partially defined goals and, in consequence, QE produced fully defined plans. However, some actions like *TurnToWP(goal)*, only modelled to a certain level of detail for the planner, were actually decomposed by the corresponding dedicated executive.

QE was able to generate solutions in few seconds. As observed in Section 4.4, the behaviours play an important role in this aspect, because they represent pre-planned valid sub-plans that contribute to improve the planner's performance. The solution obtained consisted of 12 traverse cycles, each containing 18 transitions organized in five timelines, one for each subsystem. Figure 6.14 partially illustrates how a traverse for the PR looks like.

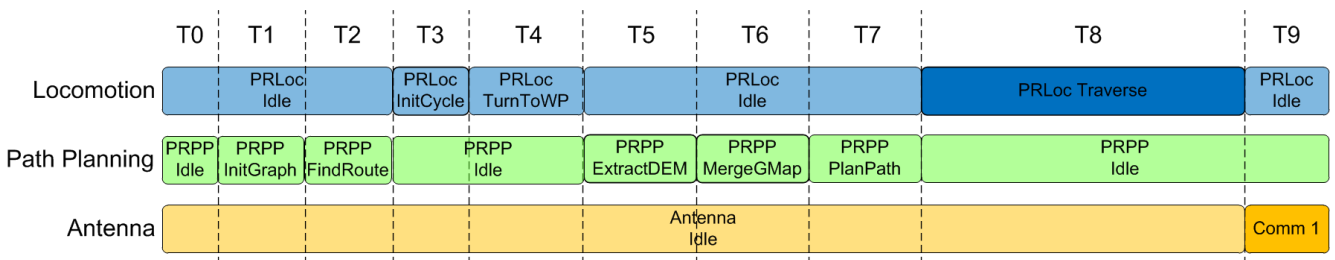


Figure 6.14.: Flexible domain timelines for the FASTER Primary Rover. The highlighted boxes represent goals.

Once the plan was uploaded in the PR OBC, all subsystems including the generic and dedicated executives were started and the flight software was commanded to start autonomous execution of the plan. At that time the role of the operators was reduced to a mere supervision activity. In case of failure, two possible repairing activities on-board were considered: (1) The executive could repair a number of specific errors without replanning; (2) Stop execution, drop the remaining activities, compute a new plan (on-ground), uplink it and launch it again.

The expected sequence of events should be the following:

-
1. Compute the shortest path to the target (crossing the hidden sand bank). SE is then commanded to start operations.
 2. The scout enters an area of loose soil and gets a “Maybe” as trafficability value. To produce a “Go” or “No-Go”, the SR deploys the penetrometer, obtaining a No-Go. An area denoted by a circle around the trap is marked as non-trafficable.
 3. *ExecutiveScoutLocomotion* then calls the failure state *SRLoc – HazardDetected* which contains a specific behaviour to repair the plan. Internally, the dedicated executive calls the path planner to recalculate the path for both rovers and traverse is resumed (see Figure 6.4). The repair activities must be executed during the time boundaries of *SRLoc – TraverseWithSS*. Otherwise, the error would be scaled up to SE and from there to QE.
 4. The scout WLSIO system identifies a “Maybe” value and deploys the mDCP obtaining a Go. The scout arrives at the first intermediate waypoint, stops and the PR starts to move towards it following the updated path. At every intermediate waypoint, the rovers are pointed towards the next one to allow the sensors to generate a map of the area they should traverse. Several perception cycles are repeated between intermediate waypoints, 4 meters apart each other.
 5. The path towards the target passes through a sand trap.
 6. The scout identifies the trap and the path is flagged as No-Go. The executive launches again the *SRLoc – HazardDetected* behaviour to repair the plan.
 7. A new path is generated to avoid the sand trap passing through a big rock.
 8. When the robots are less than four meters apart from the rock, the sensors of the SR will detect the obstacle during the *BGenerateMapTeam* activity and the path planner will produce an adequate path avoiding it (no need for repair or replan).
 9. After the avoiding manoeuvre, both robots reach the target.

As in previous test campaigns, during the final trial no re-planning was required. The first sand bank was not detected by the sensors but it did not have any consequence in the mission and therefore no intervention from the planner or executive was required. On the other hand, the executive needed to demonstrate its short-scope repair capabilities in two occasions triggered by a *SRLoc – HazardDetected* as explained before. Repair at execution level was possible thanks to an accurate modelling of complex goals and to the subsequent generation of flexible plans by the planner. In case the time assigned to complex goals had been too tight, the executive would not have had time to initiate the corresponding repair procedures, triggering as a consequence a call to the replanner.

Due to the limited time available, the rovers were stopped in nominal state before reaching the target. The overall performance of the system, and especially of the mission planner and executive, was completely successful and allowed the mission to reach the expected E3 level of autonomy, which would have been E1 otherwise.

6.4 The USAR Project

6.4.1 Description of the Mission

The aim of this project is to demonstrate high levels of autonomy beyond mere exploration in USAR (Urban Search and Rescue) environments.

The mission involves a rescue robot in a disaster scenario. The robot must first explore the area, create a map and send images of potential victims to the control center. A human operator must then verify for each picture whether or not there is a victim. Depending on the conditions, different sensors such as stereo or infrared cameras, microphones, etc. could be used. Once the robot has explored the entire area, it returns to the base station, recharge the batteries and return to the arena to conduct a patrol activity, visiting all the victims according to an optimal pre-computed path until the batteries get again depleted, moment at which it returns again to the base station and the process repeats.

Hector, displayed in Figure 6.15, is the robot used for this project. It is a four wheeled UGV equipped with a LIDAR, RGB-D camera, infrared camera, IMU and wheel/track encoders[100].

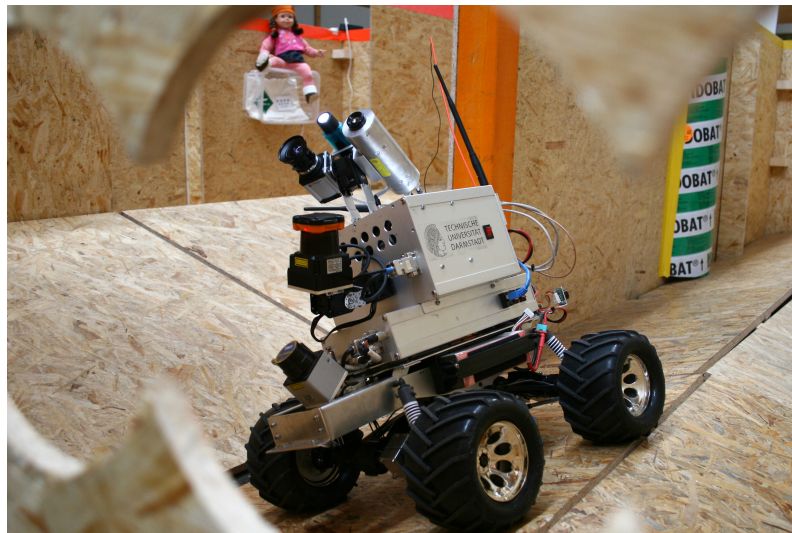


Figure 6.15.: Hector Rescue Robot (Courtesy of SIM Group, TU Darmstadt).

6.4.2 Modelling the Planner Inputs for USAR

Domain Model

The domain is constructed in two hierarchical layers: complex and primitive, with a total of five components including a battery resource. Unlike in the FASTER case, these components do not use directly any hardware functionality, but rather Hector complex behaviours.

Figure 6.16 illustrates the DDL model of Hector used by QuijoteExpress.

Mission

Abstract component containing complex goals for the rescue mission. *Explore* directly maps to the Hector behaviour *HLoc – Explore* (see below), *Patrol* starts a loop in which the robot visits all the victims found and *GoToBase* returns the robot to the base station.

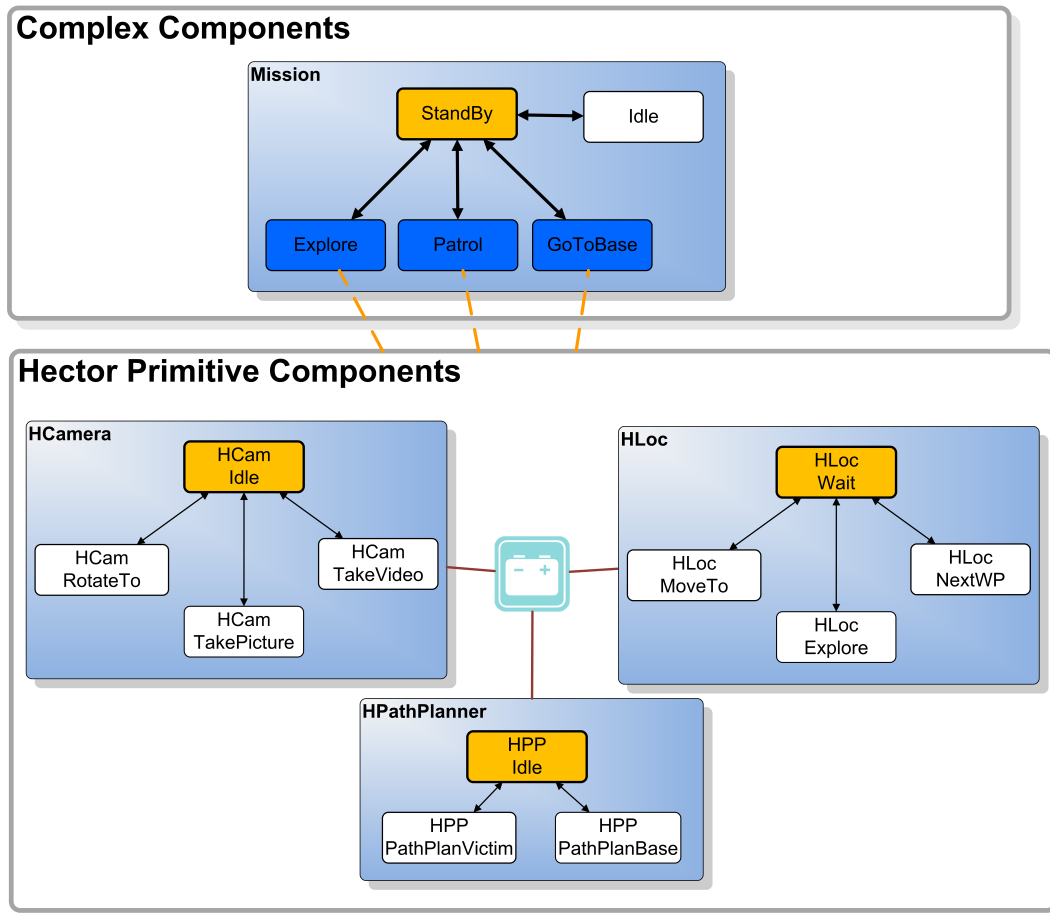


Figure 6.16.: Hector Subsystems.

Camera

Includes functionalities to orientate the mast and operate its cameras, allowing to take pictures or record videos of the targeted position.

Locomotion

Contains Hector mobility complex behaviours. *HLoc – Explore* initiates the exploration of the entire area, getting into any open space it finds and looking for victims with the cameras mounted in the mast. Every time the robot considers that it has found a victim, pictures are sent to the operator who must verify them. *HLoc – NextWP* moves the robot to the next waypoint computed by the path planner and *HLoc – MoveTo* to a specific coordinate.

Path Planner

Contains activities related to path planning. *HPPPathPlanBase* computes the path to a single waypoint represented by the base station. *HPPPathPlanVictims* is based on a Travel Salesman Problem (TSP) strategy to compute the optimal path passing by a number of waypoints, the coordinates of which correspond to the positions where the victims were found.

Behaviours (Hector DataBase)

Each complex activity of the *Mission* component is divided in a behaviour illustrated in Figure 6.17

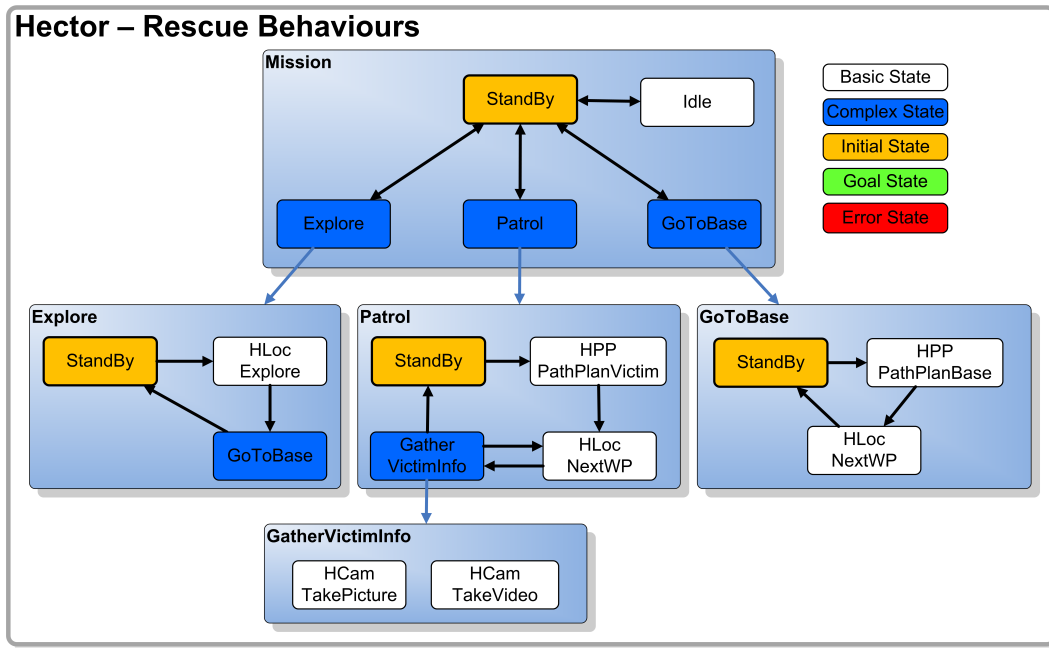


Figure 6.17.: Hierarchical behaviours included in the KDB of QuijoteExpress for USAR project.

Explore first calls the method *HLoc – Explore* described before. During the exploration, the operator is confirming each of the potential victims detected by Hector. Once the robot has explored the whole area, it returns to base. The idea is that the robot could carry some equipment to the victims.

Patrol first computes an optimal path passing by all the waypoints where a victim has been confirmed and goes to visit them. Every time Hector reaches a waypoint, it stops and gather some information (pictures and video). The idea is to assess the state of the victim and to allow the victim to provide information to the operators via videostream.

In case the battery goes below a certain level, *GoToBase* is called. This behaviour computes the path to the base and drives the robot there.

External Functions

The behaviours named before required some functionalities that were not implemented for Hector.

At the time of the test, the connection between the planner and executive was not ready. To allow *replanning*, RobCon was configured to connect to an auxiliary component that had pre-computed plans ready to be injected in the executive. More specifically, there were three pre-computed plans, one for each behaviour, which activities could be used regardless of the specific situation of the robot.

Another external function was the battery simulator, used to simulate the estimation of battery consumption. It also allowed to easily adapt the duration of the tests just by changing the ratio of consumption estimated, forcing the robot to go back to the control station.

A third external function was used to estimate the order in which the victims had to be visited by means of a TSP strategy.

6.4.3 Dedicated Executives for USAR

Three dedicated executives were implemented, one for each of the primitive components identified in the model. Implementing them was more complicated than for FASTER, as the system needed to interface with Hector legacy software.

6.4.4 Evaluation

Based on the same schema used for FASTER, the evaluation of the system was divided in a testing phase followed by a number of demonstrations with the real robot.

Testing

Testing was much easier than in FASTER for several reasons. First, unlike for FASTER, this scenario was based on consolidated software and hardware used for years in rescue competitions. Second, a very good integration with the simulator (see Figure 6.18) made it possible to test the whole mission before the demonstration. Finally, the executive had been extensively debugged during FASTER.

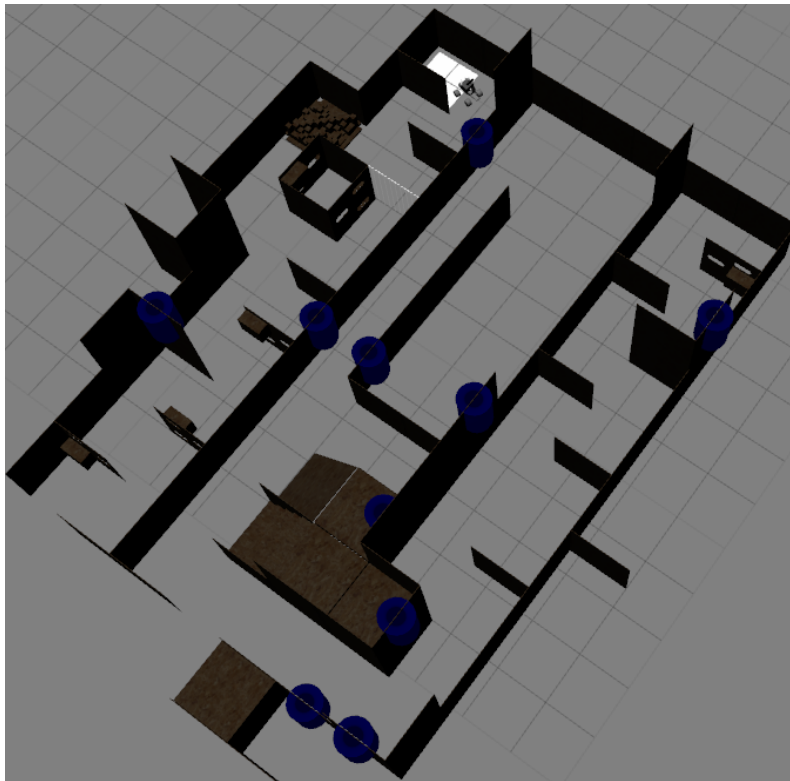


Figure 6.18.: Simulation of a rescue mission with QE and SE on-board a virtual Hector rover in the Gazebo simulator.

Demonstration

The final demonstration was run in the arena shown in Figure 6.19, used by TU Darmstadt to test software and hardware for rescue robots.

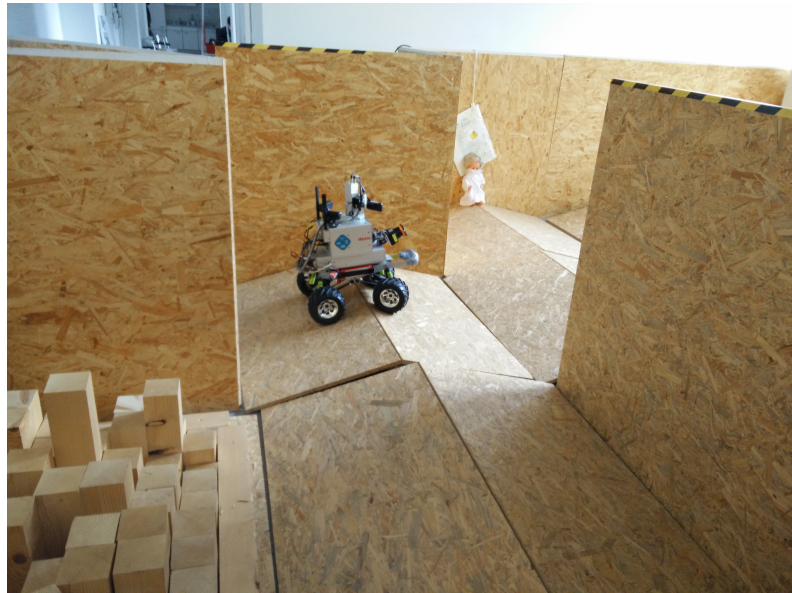


Figure 6.19.: Picture displaying Hector robot in the SIM/TU Darmstadt arena. Close to the robot there is a doll with an electric blanket representing a victim.

The mission consisted on the exploration of the whole arena, followed by a patrol activity.

```

PROBLEM Rover-Problem (DOMAIN USAR_Domain) {
//-----
// Facts
//-----
f1 <fact> compMission.t11.Idle();
f2 <fact> compCamera.t11.Idle();
f3 <fact> compLoc.t11.Idle();
f4 <fact> compPathPlanner.t11.Idle();

f1 RELEASE [0, 0];
f2 RELEASE [0, 0];
f3 RELEASE [0, 0];
f4 RELEASE [0, 0];

//-----
// Goals
//-----
g1 <goal> compMission.t11.Explore();
g2 <goal> compMission.t11.Patrol();
g1 BEFORE g2;
}

```

The aim of this scenario was to demonstrate an *E4* level of autonomy in which the robot had to discover its goals (the victims). Moreover, it allowed to test the mission planner and executive in a different scenario. RobCon also demonstrated its capability to operate at different levels of autonomy, this time connecting to a replanner instead of the user as it happened in FASTER.

Preparing the Test

The arena required almost no preparation. Three dolls were used as victims, each covered with an electric blanket in order to be detected by the infrared camera.

For the different runs, the dolls and Hector were situated in different places, some specially challenging like a small, totally dark cavity. At the start of every run the robot had the batteries fully charged.

One operator was in charge of commanding the start of the mission via a GUI (Figure 6.20). RobCon was configured to the maximum level of autonomy.

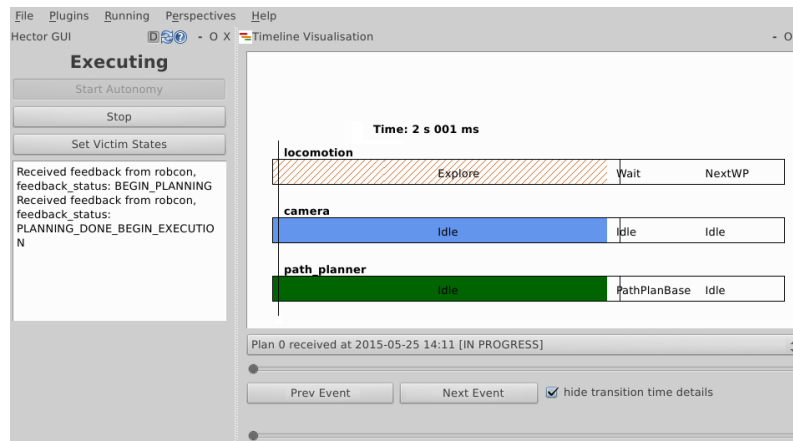


Figure 6.20.: User interface containing a representation of the timelines in execution for USAR project.

Results

Due to the simplicity of the model, consisting on just three timelines, the planner could generate plans in few milliseconds without the need of search enhancements. That makes this scenario ideal for continuous replanning due to changing conditions such as the need to recalculate the path and resources due to new obstacles blocking the way to the victims. The situation would be certainly different in case the level of control on the robot activities had been higher.

As the connection of the planner to the executive was not yet completed, three plans were pre-computed, one for each behaviour. *Explore* consisted of just three activities. *Patrol* plan consisted of an initial *HPP – PlanPathVictim* operation plus 10 loops, each containing three activities (*HLoc – NextWP*, *HCam – TakePicture* and *HCam – TakeVideo*). Finally, *GoToBase* contained three, including one single waypoint represented by the base station.

The *E4* level of autonomy was successfully demonstrated. Once the robot had discovered the victims, a new path was autonomously calculated and the robot started the *Patrol* behaviour in a loop. In addition, a type of replanning based on precomputed plans was also demonstrated. At the time the robot had virtually drained the battery, SanchoExpress stooped the execution of *Patrol* and asked *RobCon* for a new plan. *RobCon* then returned the pre-computed *GoToBase* plan which was successfully executed in all the runs.

In total, four runs with different setups were successfully completed each taking approximately 10 minutes of autonomous exploration.

6.5 Conclusions

The FASTER project was a highly representative example of the Mars rover scenario involving a realistic mission inspired in the future requirements of MSR, a real Mars rover named Bridget which is the testbed for the future ExoMars mission, and the Mars Yard, a realistic environment that simulated the conditions on Mars. The USAR project was equally representative of a disaster scenario. Hector is a state-of-the-art rescue robot that has won several prizes in the Rescue League while the arena simulates different properties of a collapsing building.

With respect to the planner, almost no time was spent on changes in the code, being most of the work dedicated to the definition of the model and behaviours, something very positive considering that modelling requires significantly less time than developing domain-dependent algorithms. Specially critical for FASTER was the selection of the actions to be performed in corner cases and the assignment of appropriate time ranges to actions and constraints in order to guarantee the executability of the plan. In USAR, the main problem consisted on properly integrating the new behaviours with those already existing for Hector.

The dedicated executives required a lot of time, specially to validate and verify the methods interfacing with the real systems. Using the executive in two different projects helped to identify and isolate common elements that were moved to the generic executive, improving the reusability of SanchoExpress.

Simulations played an important role to detect errors at early stages of the development, specially in USAR. In the case of FASTER, it was during the integration campaigns when the status of the rovers experienced the biggest evolution. For the whole first campaign and the first day of the rest of campaigns several people were required in the Mars Yard to configure the rovers. As the time passed, hardware became more reliable and people started to move to the control room to work in the software. However, it is always good to have someone in the Yard for supervision activities and fast reaction in case something goes wrong.

Getting time for testing in the rescue arena was very easy compared to the Mars Yard. As the time of the final demonstration approached, stress built up. Obtaining time to run a test became more complicated as all teams tried to get more time to check their systems. Carefully planning the tests ahead of the campaign and at the beginning of each day is crucial to maximize the benefit of scarce resources such as the Mars Yard.

Focusing on the role of the planner and executive, the expressiveness and plan robustness (in terms of flexible timelines) were the most important features of QE for the FASTER scenario, while performance was the key feature in USAR. In FASTER, an E3 level of autonomy with short-scope plan-repair was accomplished while USAR achieved an E4 level with deliberative plan-repair.

With respect to the executive, its modular approach in generic and dedicated executives helped to facilitate the integration with the different robots. In the case of FASTER, FDIR capabilities were crucial to achieve the goals of the project.

Summarizing, in this chapter *E3* and *E4* levels of autonomy have been demonstrated for two different scenarios using QE and SE. Moreover, short-scope and deliberative plan repair were demonstrated during virtual and real tests. Finally, the reusability and stability of the mission planner and executive has been proven with three different robots.



7 Conclusions

7.1 Summary

This thesis has analysed the current state of automated mission planning from a theoretical and applied point of view, identified the needs of future missions and proposed solutions in the form of a mission planner and executive which have been tested in two different scenarios. QuijoteExpress presents several novelties to the field of automated planning that can be summarized as the combination of classical and applied planning, more specifically of the following planning techniques: Timeline planning, forward-chaining and HTN. The following lines present the main ideas and how they contribute to the field of automated planning and execution.

7.1.1 HTLN

Hierarchical TimeLine Networks (**HTLN**), introduced in Chapter 3, combines Timeline and HTN planning techniques. While temporal planning is the core technique required to solve problems such as the Mars rover scenario, HTN allows the encoding of domain-specific knowledge in a natural way in the form of HTN methods that can be used by a domain-independent planner.

The **dual representation** of complex goals as hypernodes (decisions) and hypergraphs (decision networks) is one of the key features of HTLN, oriented to avoid the loss of information produced in HTN due to the replacement of complex goals by their decomposition methods. It is also one of the key differences with respect to ASPEN [43], the only known TLP planner with HTN capabilities besides QuijoteExpress.

Using HTLN, we were able to theoretically demonstrate important properties such as the soundness and completeness of the solvers, to increase the level of expressiveness from regular to context-free grammars and to improve the performance thanks to the reduction of the number of constraints derived from the use of complex goals. The capability to conduct parallel planning, derived from the analysis of certain properties of the graph representing the problem, was also introduced from a theoretical point of view.

Summarizing, the contribution of HTLN must be judged as the first theoretical formalisation of hierarchical timelines. It is also the first formalism in which task decomposition is based on an expansion rather than a replacement of a complex action by a set of subactions. Two important consequences of the formalism, the capability to perform parallel planning in multiple ways and the extraction of sufficient plans contribute to the state-of-the-art in terms of improved performance and plan robustness.

7.1.2 QuijoteExpress

A planner based on the HTLN formalism, named QuijoteExpress (QE), was introduced in Chapter 4. QE planning process is divided in two steps. A strategic solver called the Supersolver obtains the next job, which is a node of the search space, and dispatches it to one of the three (tactical) resolvers available: unfolder, decomposer and scheduler. Each resolver benefits from one of the planning techniques defined before: the unfolder is based on forward-search, the decomposer on HTN and the scheduler on TLP.

QE can perform parallel planning based on the extraction of frontiers and the subsequent division of the problem in subproblems delimited by these frontiers. The concept of frontier itself is derived from the HTLN formalism.

In addition, it inherits from classical planners such as Fast Forward [86] the concept of multi-queue, multi-heuristic evaluation and deferred heuristic evaluation, another first-in-class feature that can be exploited when the heuristics from classical planners were ported to QE.

An interesting effect derived from the combination of TLP, HTN and forward-chaining is the mix of the state-space and plan-space representations of the search space. While the unfold is based on the state-space approach, the decomposer and scheduler are plan-space. This fact is just conceptual and does not have any implications implementation level.

QE provides multiple contributions to the state-of-the-art in automated planning. It is the first real attempt to combine classical and applied planning, more specifically, the first **Timeline planner based on forward-search**. The impact of this combination in the field of applied planning is highly positive, as it facilitates the adoption of classical planning techniques. Moreover, it is the first timeline planner that exploits parallel planning to improve the performance and the first one able to produce sufficient plans to improve plan robustness.

7.1.3 SanchoExpress

A novel executive, called SanchoExpress (SE), was presented in Section 5. It is tightly integrated with the planner, with whom it shares a common plan and mission-specific knowledge in the form of HTN methods. SE is capable of performing parallel execution of flexible timelines and short-scope plan repair, two characteristics deemed crucial during the final demonstration of the FASTER project, where two rovers successfully completed a long traverse full of obstacles. SanchoExpress is, to the best of our knowledge, the first executive of its kind developed for ROS.

7.2 Future Work

While QE and SE have provided multiple answers, they also opened new questions. The following is a non-exhaustive list of topics that can be further investigated.

7.2.1 Future of Planning for Robotic Applications

Planning systems will need to be adapted to the new requirements, some of them already addressed in this thesis. QuijoteExpress is a prototype designed to be expandable, offering multiple open lines for research.

Planning Under Uncertainty

In those environments where plan generation on-ground is difficult (if not impossible) due to the high level of **uncertainty**, that is, those that cannot be fully observed, are unstructured and/or dynamic (see assumptions A1 to A3 in Section 2.2.2), several lines of research have been analysed along the thesis. First, further research in the field of **sufficient planning** will help to produce on-ground valid, partial plans that are increasingly completed on-board as new information about the environment is gathered. Future work could focus on how to automatically define the appropriate level of decomposition of actions, based for example on the temporal order of the actions: the plan would be completely defined

in the short-term, getting more abstract for actions taking place farther in time, balancing the effort required to generate a valid plan. Better **performance** in the scope of satisfying replanning would be required to promptly react to the changing conditions, specially when high levels of autonomy such as E4 and beyond are required. In this regard, a number of suggestions are provided in the APSI paragraph. In very extreme cases, robotic missions might benefit from the flexibility given by the possibility of dropping or generating goals without human intervention, a concept coined in this thesis and named **auto-goaling**. This represents an important shift in terms of operations from goal-based replanning (E4) to autonomous on-board goal generation (**E5**).

Integrated Deliberative Layer

Besides the mission planner, a robotic software architecture might contain other reasoning systems such as path planners, expert systems to evaluate science goals, etc. Integrating them would be the ultimate step to accomplish reasoning about the mission as a whole. Some initial work has been already conducted in this direction for FASTER where the mission planner requested battery and time estimations to the path planner in order to add traverse activities to the plan.

Human-Planner Collaboration

Cooperation between the deliberative layer and humans by means of **mixed-initiative** planning should be greatly enhanced in the future. The organization of models and plans in hierarchical structures and the capability to reason with abstract, **complex goals** represent a first step to facilitate the analysis, verification and validation of models and plans, which were identified as key aspects for the space sector during the recent RCOS (Robot Control Operating System) forum at ASTRA 2015.

APSI and APSI*

The work accomplished in QuijoteExpress helped to discover some limitations in APSI. Starting with the language, several requirements coming from QE such as identification of default values or decompositions have been already added. Re-designing DDL as an object oriented language will provide multiple benefits during the modelling phase, specially from the use of inheritance, a feature that Europa's NDDL language already offers[9]. With respect to performance, the major drawback of APSI is the existence of one single temporal network in the DomainManager. This approach forces the scheduler to add/retract all the decisions and relations every time a new node is created in the search space. Moreover, it prevents the use of all kinds of parallelism during scheduling with the exception of multi-heuristic evaluation. The problem lies on the need to lock the domain to guarantee the consistency of the STP network. Solving this problem and making APSI thread-safe would drastically improve the performance.

Forward-Chaining Temporal Planning

This technique could be evolved in different ways. First of all, it might eliminate the need of scheduling as used nowadays. Because actions are already ordered, scheduling responsibilities would be reduced to calculate the time intervals associated to the ordered actions and propagate the changes in the temporal network and resource consumption profiles. Regarding search, the non-linear nature of temporal problems poses big challenges to forward-chaining planning, making A* non-optimal. Anticipating decompositions as much as possible would decrease the error of the heuristics, but other techniques based on graph-search are also promising.

Search Algorithms

Multiple search algorithms can be added in a very easy way. Specially useful on-board would be SMA*, as it limits the memory consumption and Greedy A* because it can generate faster solutions during replanning. In models where the distance to the goal can be easily measured, applying IMBA* [23] based on graph-cuts could improve the performance while maintaining the optimality and completeness of A*.

Search Space

In search spaces with graph topologies, detecting duplicates becomes important. Even though QuijoteExpress does not incorporate any of these algorithms, it is possible to give some hints on the way to go. Provided that every value and every type of temporal and parametric constraint has a unique numerical identifier, it would be possible to compute the sum of all the decisions and constraints for each node generated. In case the sum of a new node were different to that of all nodes in the closed, the new node could not be a duplicate. This is a necessary but not sufficient condition, therefore the opposite case would require an exhaustive comparison of all their decisions and relations.

Heuristics

Using forward-chaining search facilitates the use of heuristics developed for classical planners. PDBs and Landmark heuristics might be highly useful to prune the search space in APSI* based planners thanks to their conceptual similarities with respect to HTLN methods. It would be interesting to compare them with HTN methods [61, 60], which might help to understand how to decrease the negative impact of decompositions in forward-chaining search.

7.2.2 Future of Execution on New Robotic Applications

With respect to the reactive layer, executing directly the Decision Network without the need of extracting the timelines would simplify the integration of the planner and executive. The framework IDEA [109] already explored this option but from an architectural rather than planning point of view.

SanchoExpress has been developed for ROS. It is uncertain whether this system will be ready in the future for space applications where RAMS (Reliability, Availability, Maintainability and Safety) systems are crucial. While SE can be used “as is” on Earth, space applications might require the concept to be adapted to other platforms.

7.3 Closing Remarks

The robot industry is experiencing a very rapid growth led by an increasing demand for both service and industrial sectors. The trigger of the *robot revolution* will be the massive transition from structured environments such as factories to the open world, where robotic systems are expected to be used in diverse scenarios such as healthcare, transport & logistics, defense, homecare or emergency response. This transition will demand multiple innovations in different areas, being artificial intelligence, and more specifically decision-making technologies one of the most relevant. Competitions such as the Darpa Robotic Challenge or the RoboCup, the decreasing price of hardware and the irruption of de-facto open source standards such as ROS, used by millions of developers, have re-shaped the state-of-the-art, accelerating the evolution of ground robots.

With a much smaller market, space robotics would certainly benefit from the technologies tested on Earth in future missions such as Mars Sample Return or a hypothetical return to the Moon, in which

astronauts are expected to closely collaborate with exploration robots. In this regard, the Horizon 2020 programme has identified six technologies, including autonomy, that will possibly become full scale applications in the 2025-2035 decade. This thesis makes several contributions in the fields of automated planning and execution, two technologies that will be increasingly demanded on-board space robotic missions.





A Appendices

A.1 HTLN nomenclature

$v, v^p, v^c, V^p, V^c, v^{ref}, V^{all}$	Value, Primitive value, Complex value, Reference (complex) value of a method, Set of primitive values, Set of complex values, Set of all values
$d_v = \langle \tau_d, v \rangle, d_v^p, d_v^c, d_v^{ref}, d^+, d^-, D^p, D^c, D^{all}$	Decision where $\tau_d \in H$ is the time frame of the decision and $v \in V^c, v \in N$ a state variable value, Primitive decision, Complex decision, Reference (complex) decision of an instantiated method decision network, Decisions to be added, Decisions to be deleted, Set of primitive decisions, Set of complex decisions, Set of all decisions
$d_{i_{front}}^{tl}, d_{g_{front}}^{tl}$	Decision frontier for the timeline tl in the initial and goal state respectively
$pred_v$ $\langle s_v \in S, X_v = \langle x_1^v, \dots, x_m^v \rangle \rangle$	= Predicate of the value v , where the symbol s_v represents the name of the value and X_v is the set of parameters of v
$f, f^+, f^-, f_{temp}^{typet}(t_1, t_2), f_{param}^{typet}(t_1, t_2, \dots, t_m)$	Relation, Set of relations to be added, Set of relations to be deleted, Temporal relation, Parameter relation
\mathbb{D}	Domain of a variable
$m = (v_m, dn_m), M$	Method where v_m is the reference value and dn_m the target network, Set of all methods of a domain
$dn = \langle N, E \rangle, dn_i, dn_m, dn_{d_v}^{dec}$	Decision Network with a set N of nodes and a set E of edges, i-th decision network, method decision network, Instantiated target network of a method which reference value is v
P	Problem
D	Domain
$\sigma = \langle T_\sigma, N_\sigma \rangle$	Behaviour, where T_σ is a finite set of ordered time instants in H and N_σ is an assignment of values to the time instants in T_σ
n, n_{m_i}, N	Node, i-th node of the target network of method m , Set of all nodes of a graph/hypergraph
$e_i, e_{m_i}, E_{modif}, E_{param}, E_{temp}, E$	Edge, i-th edge of m 's target network, Set of modified relations of resulting of the instantiation of m 's target network, Set of additional parameter relations resulting of the instantiation of m 's target network, Set of additional temporal relations resulting of the instantiation of m 's target network, Set of all edges of a graph/hypergraph
$\rho(D, dn_i, dn_{goal}), \rho_v, \rho_\sigma, \rho_\delta, \rho_\chi, \mathbb{P}$	General resolver which inputs are the domain, current network and goal network; Unfolder; Scheduler; Decomposer; Timeline completer; Set of all resolvers
$\theta(d_v, dn_m)$	Most general unifier of a decision d_v and a method target network dn_m
$\rho_\delta(dn_i, d_v, dn_m, \theta)$	Decomposition in dn_i of value d_v in the target network dn_m under the unification θ
$\gamma(dn_i, \rho)$	Transition function of a network dn_i after the application of the output of a resolver ρ
C, C_{SV}, C_C, C_R	Set of components of a domain, state variable components, consumable resource components, reusable resource components
$\tau_i = [t_i, t_{i+1})$	Time interval starting in t_i (inclusive) and ending in t_{i+1} (excluded)

- [1] T. D. Ahlstrom, M. E. Diftler, R. B. Berka, J. M. Badger, S. Yayathi, A. W. Curtis, and C. A. Joyce. Robonaut 2 on the international space station: Status update and preparations for iva mobility. Technical report, NASA Johnson Space Center, 2013.
- [2] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, 17:315–337, 1998.
- [3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, Nov. 1983.
- [4] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1-2):123 – 191, Jan. 2000.
- [5] F. Bacchus and Q. Yang. Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, 71:43–100, 1993.
- [6] M. Bajracharya, M. W. Maimone, and D. Helmick. Autonomy for mars rovers: Past, present, and future. *IEEE Computer*, 41:44–50, 2008.
- [7] M. Bartusch, R. Möhring, and F. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, 1988.
- [8] C. Bäckström and B. Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–655, 1996.
- [9] T. Bedrax-Weiss, C. McGann, and A. Bachmann. Europa2: User and contributor guide. Technical report, NASA Ames Research Center, 2005.
- [10] J. Benton, A. J. Coles, and A. I. Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Twenty Second International Conference on Automated Planning and Scheduling (ICAPS-12)*, June 2012.
- [11] C. Berge. Optimisation and hypergraph theory. *European Journal of Operational Research*, 46(3):297–303, June 1990.
- [12] A. Berthoz. *The Brain’s Sense of Movement*. Perspectives in Cognitive Neuroscience Series. Harvard University Press, 2002.
- [13] P. Bertoli, A. Cimatti, M. Roveri, and 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.
- [14] C. Bessière. Arc-consistency and arc-consistency again. *Artif. Intell.*, 65:179–190, January 1994.
- [15] J. Biatek, M. W. Whalen, M. P. E. Heimdahl, S. Rayadurgam, and M. R. Lowry. Analysis and testing of plexil plans. In *Proceedings of the 2Nd FME Workshop on Formal Methods in Software Engineering, FormaliSE 2014*, pages 52–58, New York, NY, USA, 2014. ACM.
- [16] J. J. Biesiadecki, C. Leger, and M. W. Maimone. Tradeoffs between directed and autonomous driving on the mars exploration rovers. *The International Journal of Robotics Research*, 26(1):91–104, Jan. 2007.
- [17] J. J. Biesiadecki and M. W. Maimone. The mars exploration rover surface mobility flight software driving ambition. In *IEEE Aerospace Conference (IAC)*, 2006.
- [18] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. P. Miller, and M. G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2):237–256, Apr. 1997.
- [19] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

-
- [20] J. L. Bresina, A. K. Jónsson, P. H. Morris, and K. Rajan. Activity planning for the mars exploration rovers. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 40–49, 2005.
- [21] J. L. Bresina, A. K. Jonsson, P. H. Morris, and K. Rajan. Mixed-initiative activity planning for mars rovers. *Challenges*, pages 2–3, 2003.
- [22] J. L. Bresina, A. K. Jonsson, P. H. Morris, and K. Rajan. Mixed-initiative planning in mapgen: Capabilities and shortcomings. In *Proceedings of the Int. Conf. on Planning and Scheduling (ICAPS) Workshop on Mixed-Initiative Planning and Scheduling*, page 8, 2005.
- [23] D. Burkett, D. Hall, and D. Klein. Optimal graph search with iterated graph cuts. In *AAAI*, 2011.
- [24] M. H. Burstein, B. Beranek, N. Inc, and D. V. Mcdermott. Issues in the development of human-computer mixed-initiative planning. In *Cognitive Technology*, pages 285–303. Elsevier, 1996.
- [25] J. Carsten, A. Rankin, D. Ferguson, and A. Stentz. Global planning on the mars exploration rovers: Software integration and surface testing. *Journal of Field Robotics*, 26(4):337–357, 2009.
- [26] G. Casonato and G. Palmerini. Visual techniques applied to the atv/Iss rendezvous monitoring. In *Aerospace Conference, 2004. Proceedings. 2004 IEEE*, volume 1, pages –625 Vol.1, March 2004.
- [27] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI’94, pages 1023–1028, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.
- [28] A. Castano, A. Fukunaga, J. Biesiadecki, L. Neakrase, P. Whelley, R. Greeley, M. Lemmon, R. Castano, and S. Chien. Automatic detection of dust devils and clouds on mars. *Machine Vision and Applications*, 19(5-6):467–482, 2008.
- [29] R. Castaño, T. A. Estlin, R. C. Anderson, D. M. Gaines, A. Castano, B. J. Bornstein, C. Chouinard, and M. Judd. Oasis: Onboard autonomous science investigation system for opportunistic rover science. *J. Field Robotics*, 24(5):379–397, 2007.
- [30] A. Ceballos, S. Bensalem, A. Cesta, L. S Silva, S. Fratini, F. Ingrand, J. Ocon, A. Orlandini, F. Py, K. Rajan, R. Rasconi, and M. V. Winnendael. A goal-oriented autonomous controller for space exploration. *ASTRA*, 2011.
- [31] A. Cesta, G. Cortellessa, M. Denis, A. Donati, S. Fratini, A. Oddi, N. Policella, E. Rabenau, and J. Schulster. Raxem - supporting command uplink in mars express. In *Proceedings of the 9th International Symposium in Artificial Intelligence, Robotics and Automation in Space, iSAIRAS-08*, 2008.
- [32] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Mexar2: Ai solves mission planner problems. *IEEE Intelligent Systems*, 22(4):12–19, 2007.
- [33] A. Cesta, G. Cortellessa, S. Fratini, and A. Oddi. Developing an end-to-end planning application from a timeline representation framework. In K. Z. Haigh and N. Rychtycky, editors, *IAAI. AAAI*, 2009.
- [34] A. Cesta, G. Cortellessa, S. Fratini, A. Oddi, and N. Policella. Software companion the mexar2 support to space mission planners. In *Proceeding of the 2006 conference on ECAI 2006*, pages 622–626, Amsterdam, The Netherlands, The Netherlands, 2006. IOS Press.
- [35] A. Cesta and S. Fratini. The timeline representation framework as a planning and scheduling software development environment. In *27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-08)*, Edinburgh, UK, 2008.
- [36] A. Cesta, S. Fratini, A. Orlandini, and R. Rasconi. Continuous planning and execution with timelines. In *International Symposium on Artificial Intelligence (i-SAIRAS)*, 2012.
- [37] A. Cesta, A. Oddi, and S. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2002.
- [38] Y. Chen, B. W. Wah, and C.-W. Hsu. Temporal planning using subgoal partitioning and resolution in sgplan. *J. Artif. Int. Res.*, 26(1):323–369, Aug. 2006.

-
- [39] S. Chien, M. Johnston, J. D. Frank, M. Giuliano, A. Kavelaars, C. Lenzen, and N. Policella. A generalized timeline representation, services, and interface for automating space mission operations. In *12th International Conference on Space Operations (SpaceOps)*, 2012.
- [40] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve responsiveness of planning and scheduling. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling*, pages 300–307, 2000.
- [41] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Integrated planning and execution for autonomous spacecraft. *Aerospace and Electronic Systems Magazine, IEEE*, 24(1):23–30, Jan 2009.
- [42] S. Chien, G. Rabideau, R. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. Estlin, B. Smith, F. W. Fisher, T. Barrett, G. Stebbins, and D. Tran. Aspen - automated planning and scheduling for space mission operations. *Jet Propulsion*, pages 1–10, 1997.
- [43] S. Chien, G. R. Rabideau, R. L. Knight, R. Sherwood, B. Engelhardt, D. Mutz, T. A. Estlin, B. Smith, F. W. Fisher, A. C. Barrett, G. Stebbins, and D. Tran. Aspen - automated planning and scheduling for space mission operations. In *International Conference on Space Operations (SpaceOps)*, pages 1–10, 2000.
- [44] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castaño, A. Davies, D. Mandl, S. Frye, B. Trout, J. D’Agostino, S. Shulman, D. Boyer, S. Hayden, A. Sweet, and S. Christa. Lessons learned from autonomous sciencecraft experiment. In *Proceedings of the fourth international joint conference on Autonomous agents and multi-agent systems*, AAMAS ’05, pages 11–18, New York, NY, USA, 2005. ACM.
- [45] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davies, R. Lee, D. Mandl, S. Frye, B. Trout, J. Hengemihle, J. D’Agostino, S. Shulman, S. Ungar, T. Brakke, D. Boyer, J. Van Gaasbeck, R. Greeley, T. Doggett, V. Baker, J. Dohm, and F. Ip. The eo-1 autonomous science agent. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS ’04, pages 420–427, Washington, DC, USA, 2004. IEEE Computer Society.
- [46] A. Chmeiss and P. Jegou. Two new constraint propagation algorithms requiring small space complexity. In *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, pages 286–289, Nov 1996.
- [47] E. Clarke, D. Long, and K. McMillan. Compositional model checking. In *Logic in Computer Science, 1989. LICS ’89, Proceedings., Fourth Annual Symposium on*, pages 353–362, jun 1989.
- [48] B. J. Clement and E. H. Durfee. Exploiting domain knowledge with a concurrent hierarchical planner. In *AIPS Workshop on Analysing and Exploiting Domain Knowledge for Efficient Planning*, 2000.
- [49] B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract reasoning for planning and coordination. *JOURNAL OF AI RESEARCH*, 28:453–515, 2007.
- [50] A. Coles, A. Coles, M. Fox, and D. Long. Forward-chaining partial-order planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.
- [51] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Popf2: a forward-chaining partial order planner. In *Procs. IPC-7*, 2011.
- [52] M. L. Commons. Introduction to the model of hierarchical complexity and its relationship to postformal action. *World Futures: The Journal of General Evolution*, 64(5-7):305–320, June 2008.
- [53] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, May 1991.
- [54] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, December 1987.
- [55] M. Diftler, R. Ambrose, S. M. Goza, K. S. Tyree, and E. Huber. Robonaut mobile autonomy: Initial experiments. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1425–1430, April 2005.
- [56] M. Drummond. On precondition achievement and the computational economics of automatic planning. In *European Workshop on Planning systems*, 1994.

-
- [57] D. L. D. Dvorak. Nasa study on flight software complexity. Technical report, NASA, 2009.
- [58] S. Edelkamp. Planning with pattern databases. In *European Conference On Planning (ECP-01)*, pages 13–24, 2001.
- [59] S. Edelkamp and J. Hoffmann. Pddl2. 2: The language for the classical part of the 4th international planning competition. Technical report, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2004.
- [60] M. Elkawagy, P. Bercher, B. Schattenberg, and S. Biundo. Improving hierarchical planning performance by the use of landmarks. In *Proceedings of the 26th National Conference on Artificial Intelligence (AAAI 2012)*, pages 1763–1769. AAAI Press, 7 2012.
- [61] M. Elkawagy, B. Schattenberg, and S. Biundo. Landmarks in hierarchical planning. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 229–234. IOS Press, 2010.
- [62] K. Erol, J. Hendler, and D. S. Nau. Complexity results for htn planning. *Annals of Mathematics and Artificial Intelligence*, 18:69–93, 1994.
- [63] K. Erol, J. Hendler, and D. S. Nau. Umcp: A sound and complete procedure for hierarchical task-network planning. *Artificial Intelligence Planning Systems (AIPS)*, pages 249–254, 1994.
- [64] K. Erol, J. A. Hendler, and D. S. Nau. Htn planning: Complexity and expressivity. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 2.*, pages 1123–1128, 1994.
- [65] T. Estlin, R. C. Anderson, D. Blaney, M. Burl, B. Bornstein, R. Castano, L. De Flores, D. Gaines, M. Judd, D. R. Thompson, and R. Wiens. Aegis automated targeting for the msl chemcam instrument, 2013.
- [66] T. Estlin, D. Gaines, F. Fisher, and R. Castano. Coordinating multiple rovers with interdependent science objectives. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05*, pages 879–886, New York, NY, USA, 2005. ACM.
- [67] T. A. Estlin, B. J. Bornstein, D. Gaines, R. C. Anderson, D. R. Thompson, M. Burl, R. Castano, and M. Judd. Aegis automated science targeting for the mer opportunity rover. *International Symposium on Artificial Intelligence Robotics and Automation in Space (i-SAIRAS)*, pages 1–25, 2010.
- [68] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence, IJCAI'71*, pages 608–620, San Francisco, CA, USA, 1971. Morgan Kaufmann Publishers Inc.
- [69] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res.*, 20(1):61–124, Dec. 2003.
- [70] J. D. Frank. What is a timeline? In *4th Workshops on Knowledge Engineering for Planning and Scheduling at ICAPS-13*, 2013.
- [71] J. D. Frank and A. K. Jonsson. Constraint-based attribute and interval planning. *Journal of Constraints Special Issue on Constraints and Planning*, 8(4):339–364, 2003.
- [72] S. Fratini and A. Cesta. The apsi framework: A platform for timeline synthesis. *Proceedings of the 1st Workshops on Planning and Scheduling with Timelines PSTL-12*, 2012.
- [73] S. Fratini, A. Cesta, A. Orlandini, R. Rasconi, and R. D. Benedictis. Apsi-based deliberation in goal oriented autonomous controllers. In *In Proc. of 11th ESA Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2011.
- [74] S. Fratini, F. Pecora, and A. Cesta. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.
- [75] A. S. Fukunaga, G. Rabideau, S. Chien, and D. Yan. Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space*, 1997.

-
- [76] M. Gallien, F. Ingrand, and S. Lemaï. Robot actions planning and execution control for autonomous exploration rovers. In *International Conference on Automated Planning & Scheduling (ICAPS'14) Workshop on Plan Execution*, 2005.
- [77] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2-3):177 – 201, 1993.
- [78] E. Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. Technical report, AAAI, 1996.
- [79] B. Gerkey. Why ros 2.0.
- [80] M. Ghallab and H. Laruelle. Representation and control in ixtet, a temporal planner. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, pages 61–67, 1994.
- [81] M. Ghallab, D. S. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004.
- [82] E. Guizzo. Fukushima robot operator writes tell-all blog. *IEEE Spectrum*, 2011.
- [83] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(August):1–28, 1992.
- [84] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 140–149, 2000.
- [85] S. Hayati, A. Rankin, W. Kim, P. Leger, S. Chien, and A. Khaled. Advanced robotics technology infusion to the nasa mars exploration rover (mer) project. In *IFAC Symposium on Intelligent Autonomous Vehicles*, 2007.
- [86] M. Helmert. The fast downward planning system. *J. Artif. Int. Res.*, 26(1):191–246, July 2006.
- [87] M. Helmert and C. Domshlak. Landmarks, critical paths and abstractions: What’s the difference anyway? In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.
- [88] M. Helmert, P. Haslum, and J. Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, pages 176–183, 2007.
- [89] M. Helmert and H. Lasinger. The scanalyzer domain: Greenhouse logistics as a planning problem. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2010.
- [90] M. Helmert and S. Richter. Fast downward-making use of causal dependencies in the problem representation. In *Proc. IPC4, ICAPS*, pages 41–43, 2004.
- [91] J. Hoffmann and B. Nebel. The ff planning system: Fast plan generation through heuristic search. *J. Artif. Int. Res.*, 14(1):253–302, May 2001.
- [92] J. Hoffmann, J. Porteous, and L. Sebastia. Ordered landmarks in planning. *J. Artif. Int. Res.*, 22(1):215–278, Nov. 2004.
- [93] A. K. Jonsson, P. H. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space: Theory and practice. *Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, 2000.
- [94] S. Kambhampati. Comparing partial order planning and task reduction planning: A preliminary report, 1994.
- [95] S. Kambhampati. A comparative analysis of partial order planning and task reduction planning. *SIGART Bull.*, 6(1):16–25, Jan. 1995.
- [96] S. Kambhampati and J. A. Hendler. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3):193–258, June 1992.
- [97] E. Karpas and C. Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI'09*, pages 1728–1733, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

-
- [98] W. S. Kim, R. D. Steele, A. I. Ansar, A. Khaled, and I. Nesnas. Rover-based visual target tracking validation and mission infusion. In *AIAA Space Conference*, 2005.
- [99] C. Knoblock. Automatically generating abstractions for planning. *Artificial Intelligence*, 68:243–302, 1994.
- [100] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, O. von Stryk, and U. Klingauf. Robocuprescue 2014 - robot league team hector darmstadt (germany). Technical report, Technische Universität Darmstadt, 2014.
- [101] S. LEMAI-CHENEVIER. *IXTET-EXEC: Planning, plan repair and execution control with time and resource management*. PhD thesis, Ecole Doctorale Informatique et Telecommunications, 2004.
- [102] N. Lipovetzky. Visit-all domain description paper, 2010.
- [103] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99 – 118, 1977.
- [104] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,, 1998.
- [105] A. Merlo, J. Larranaga, and P. Falkner. Sample fetching rover (sfr) for msr. In *ASTRA Conference*, 2011.
- [106] A. H. Mishkin, D. Limonadi, S. L. Laubach, and D. S. Bass. Working the martian night shift. *Odyssey*, 2006.
- [107] J. R. Morris, M. D. Ingham, A. H. Mishkin, R. D. Rasmussen, and T. W. Starbird. Application of state analysis and goal-based operations to a mer mission scenario. In *Proceedings of SpaceOps Conference, Rome, Italy*, page 12, 2006.
- [108] N. Muscettola. Hsts: Integrating planning and scheduling. In M. Zweben and M. Fox, editors, *Intelligent Scheduling*. Morgan Kauffmann, 1994.
- [109] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt. Idea: Planning at the core of autonomous reactive agents. In *in Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, 2002.
- [110] N. Muscettola, C. Fry, K. Rajan, B. Smith, S. Chien, G. Rabideau, and D. Yan. On-board planning for new millennium deep space one autonomy. *1997 IEEE Aerospace Conference*, pages 303–318, 1997.
- [111] N. Muscettola, S. F. Smith, A. Cesta, and D. D’Aloise. Coordinating space telescope operations in an integrated planning and scheduling architecture, 1992.
- [112] N. Museettola, M. S. Fox, and M. Zweben. Hsts: Integrating planning and scheduling, 1993.
- [113] K. Nagatani, S. Kiribayashi, Y. Okada, K. Otake, K. Yoshida, S. Tadokoro, T. Nishimura, T. Yoshida, E. Koyanagi, M. Fukushima, and S. Kawatsuma. Emergency response to the nuclear accident at the fukushima daiichi nuclear power plants using mobile rescue robots. *J. Field Robot.*, 30(1):44–63, Jan. 2013.
- [114] D. Nau, T.-C. Au, O. Ilghami, U. Kuter, D. Wu, F. Yaman, H. Munoz-Avila, and J. Murdock. Applications of shop and shop2. *Intelligent Systems, IEEE*, 20(2):34–41, March 2005.
- [115] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz Avila. Shop: Simple hierarchical ordered planner. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.
- [116] D. S. Nau, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:379–404, 2003.
- [117] D. S. Nau, H. Muñoz Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.
- [118] A. Ogilvie, J. Allport, M. Hannah, and J. Lymer. Autonomous satellite servicing using the orbital express demonstration manipulator system. In *International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS’08)*, 2008.
- [119] E. C. on Space Standardization (ECSS). Ecss-e-70-11 space segment operability.

-
- [120] M. E. Pollack, D. Joslin, and M. Paolucci. Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research Intelligence*, 6:223–262, 1997.
- [121] R. Prakash, P. Burkhart, A. Chen, K. Comeaux, C. Guernsey, D. Kipp, L. Lorenzoni, G. Mendeck, R. Powell, T. Rivellini, A. San Martin, S. Sell, A. Steltzner, and D. Way. Mars science laboratory entry, descent, and landing system overview. In *Aerospace Conference, 2008 IEEE*, pages 1–18, March 2008.
- [122] F. Py, K. Rajan, and C. McGann. A systematic agent framework for situated autonomous systems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 2 - Volume 2, AAMAS '10*, pages 583–590, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.
- [123] G. Rabideau, B. Engelhardt, and S. A. Chien. Using generic preferences to incrementally improve plan quality. In S. Chien, S. Kambhampati, and C. A. Knoblock, editors, *AIPS*, pages 236–245. AAAI, 2000.
- [124] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, and A. Govindjee. Iterative repair planning for spacecraft operations using the aspen system. *Symposium A Quarterly Journal In Modern Foreign Literatures*, 1999.
- [125] K. Rajan, F. Py, C. McGann, J. Ryan, T. Oâ€™Reilly, T. Maughan, and B. Roman. Onboard adaptive control of auvs using automated planning and execution. *International Symposium on Unmanned Untethered Submersible Technology (UUST)*, 2009.
- [126] S. Richter and M. Helmert. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, 2009.
- [127] S. Richter, M. Helmert, and M. Westphal. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 975–982, 2008.
- [128] S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *CoRR*, abs/1401.3839, 2014.
- [129] S. Richter. *Landmark-Based Heuristics and Search Control for Automated Planning*. PhD thesis, Institute of Intelligent and Integrated Systems Faculty of Science, Environment, Engineering and Technology Griffith University, Brisbane, Australia, 2010.
- [130] J. Rintanen. Heuristics for planning with sat. In *International Conference on Principles and Practice of Constraint Programming*, 2010.
- [131] G. Röger and M. Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pages 246–249, 2010.
- [132] R. D. Rugg. Building a hypergraph-based data structure. *AUTOCARTO 6*, 2:211–220, 1983.
- [133] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.
- [134] E. D. Sacerdoti. *A Structure for Plans and Behavior*. PhD thesis, Stanford University, Stanford, CA, USA, 1977. AAI7605794.
- [135] P. Schenker. Advances in rover technology for space exploration. In *Aerospace Conference, 2006 IEEE*, pages 23 pp.–, 2006.
- [136] R. Sherwood, S. Chien, D. Tran, A. Davies, R. Castano, and G. Rabideau. Enhancing science and automating operations using onboard autonomy. In *International Conference on Space Operations (SpaceOps)*, volume 1, page 14, 2006.
- [137] R. Sherwood, A. Govindjee, D. Yan, G. R. Rabideau, S. Chien, and A. Fukunaga. Using aspen to automate eo-1 activity planning. In *IEEE Aerospace Conference*, pages 145–152, 1998.

-
- [138] N. Silva, R. Lancaster, and J. Clemment. Exomars rover vehicle mobility functional architecture and key design drivers. In *Astra Conference 2013*, 2013.
- [139] F. Simone. Apsi timeline representation framework v. 3.0. Technical report, ESA, 2014.
- [140] M. Singh. Path consistency revisited. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, TAI '95, pages 318–, Washington, DC, USA, 1995. IEEE Computer Society.
- [141] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.
- [142] D. E. Smith. The mystery talk. In *International Conference on Artificial Intelligence Planning and Scheduling Systems*, 2000.
- [143] D. Stormont. Autonomous rescue robot swarms for first responders. In *Computational Intelligence for Homeland Security and Personal Safety, 2005. CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on*, pages 151–157, March 2005.
- [144] A. Tate. Generating project networks. In *Proceedings of IJCAI-77*, pages 888–893, 1977.
- [145] A. Tate, B. Drabble, and R. Kirby. O-plan2: an open architecture for command, planning and control. *Intelligent Scheduling*, 1:213–239, 1994.
- [146] J. D. Tenenbergh. *Abstraction in Planning*, chapter 4, pages 213–288. Morgan Kaufmann, 1991.
- [147] F. Teston and R. Creasey. Proba: Esa's autonomy and technology demonstration mission. In *International Astronautical Congress*, 1997.
- [148] R. Tsuneto, J. Hendler, and D. S. Nau. Analyzing external conditions to improve the efficiency of htn planning. *IN SIXTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 913–920, 1998.
- [149] R. Tsuneto, D. S. Nau, and J. Hendler. Plan-refinement strategies and search-space size. In *PROCEEDINGS OF THE EUROPEAN CONFERENCE ON PLANNING*, pages 414–426, 1997.
- [150] M. Vallati, L. Chrupa, and T. L. McCluskey. The 2014 international planning competition. Technical report, University of Huddersfield, 2014.
- [151] V. Verma, A. Jonsson, C. Pasareanu, and M. Latauro. Universal executive and plexil: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space 2006 Conference*, 2006.
- [152] V. Verma, A. Jónsson, R. Simmons, T. Estlin, and R. Levinson. Survey of command execution systems for nasa spacecraft and robots. *Plan Execution: A Reality Check. Workshop at The International Conference on Automated Planning & Scheduling, ICAPS*, pages 92–99, 2005.
- [153] M. Vilain, H. Kautz, and P. Beek. Constraint propagation algorithms for temporal reasoning. In *Readings in Qualitative Reasoning about Physical Systems*, pages 377–382. Morgan Kaufmann, 1986.
- [154] R. Volpe. Rover technology development and mission infusion beyond mer. In *Aerospace Conference, 2005 IEEE*, pages 971–981, March 2005.
- [155] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The claraty architecture for robotic autonomy. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 1, pages 1/121–1/132 vol.1, 2001.
- [156] R. Washington, K. Golden, J. Bresina, D. Smith, C. Anderson, and T. Smith. Autonomous rovers for mars exploration. In *Aerospace Conference, 1999. Proceedings. 1999 IEEE*, volume 1, pages 237–251 vol.1, 1999.
- [157] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.
- [158] D. E. Wilkins. Can ai planners solve practical problems? *Computational Intelligence*, 6(4):232–246, 1990.
- [159] D. E. Wilkins and M. desJardins. A call for knowledge-based planning. *AI MAGAZINE*, 22:99–115, 2000.

-
- [160] D. E. Wilkins, K. L. Myers, J. D. Lowrance, and L. P. Wesley. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental Theoretical Artificial Intelligence*, 7(1):121–152, 1995.
- [161] J. Willis, G. Rabideau, and C. Wilklow. The citizen explorer design and operations planning system. In *Aerospace Conference, 1999. Proceedings. 1999 IEEE*, volume 2, pages 227–234 vol.2, 1999.
- [162] M. V. Winnendael. Key aspects of onboard software and ground control of planetary rovers. ESA Internal Report, 2006.
- [163] M. J. Woods, D. Long, R. Aylett, L. Baldwin, and G. Wilson. Mars mission on-board planner and scheduler - summary report. Technical report, Scisys, 2006.
- [164] M. J. Woods, D. Long, L. Baldwin, R. Aylett, and G. Wilson. On-board planning and scheduling for the exomars mission. In *DASIA 2006: DATA Systems In Aerospace*, page 7, 2006.
- [165] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6(1):12–24, 1990.
- [166] M. Zweben, E. Davis, B. Daun, and M. Deale. Scheduling and rescheduling with iterative repair. *Systems, Man and Cybernetics, IEEE Transactions on*, 23(6):1588–1596, Nov 1993.



Own Publications

Journal Papers

Juan M. Delfa Victoria, Fratini Simone, Policella Nicola, O. von Stryk, Y. Gao, and A. Donati. Planning mars rovers with hierarchical timeline networks. *Acta Futura*, 9(9):21–29, 2014.

Conference Papers

Juan M. Delfa Victoria, Nicola Policella, Marc Gallant, Alessandro Donati, Reinhold Bertrand, Oskar von Stryk, and Yang Gao. Roben: Introducing a benchmarking tool for planetary rover planning & scheduling algorithms. In *Proceedings of the 12th International Conference on Space Operations (Spaceops)*, 2012.

Juan M. Delfa Victoria, Nicola Policella, Yang Gao, and Oskar Von Stryk. Planning mars rovers with hierarchical timeline networks. In *Proceedings of the 23rd International Joint Conferences on Artificial Intelligence (IJCAI) - AI in Space*, 2013.

Juan M. Delfa Victoria, Nicola Policella, Yang Gao, and Oskar Von Stryk. Design concepts for a new temporal planning paradigm. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS) - Planning & Scheduling for Timelines (PSTL)*, 2012.

Workshop Papers

E. Allouis, R. Marc, J. Gancet, Y.Nevatia(2), F.Cantori, R.U Sonsalla, M. Fritsche, J. Machowinski, T. Vogele, F.Comin, W. Lewinger, B. Yeomans, C. Saaj, Y.Gao, J.Delfa, P. Weclowski, K. Skocki, B. Imhof, S. Ransom, and L. Richter. Fp7 faster project - demonstration of multi-platform operation for safer planetary traverses. In *Proceedings of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2015.

Juan M. Delfa Victoria, Yang Gao, Nicola Policella, and Alessandro Donati. A domain-independent pattern recognition system to support space mission planning. In *Proceedings of the 11th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2011.

Juan M. Delfa Victoria, N Policella, M. Gallant, O. von Stryk, A. Donati, and Gao. Y. Metrics for planetary rover planning & scheduling algorithms. In *Proceedings of the 11th Workshop on Performance Metrics for Intelligent Systems (PerMIS12)*, 2012.

Juan M. Delfa Victoria, Nicola Policella, Yang Gao, and Oskar Von Stryk. Quijoteexpress - a novel apsi planning system for future space robotic missions. In *Proceedings of the 12th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2013.

Juan M. Delfa Victoria, Brian Yeomans, Y. Gao, and O. von Stryk. Testing full autonomy in a mars scenario with two collaborative rovers. In *Proceedings of the 13th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2015.



Erklärung¹

Hiermit erkläre ich, dass ich die vorliegende Arbeit, mit Ausnahme der ausdrücklich genannten Hilfsmittel, selbständig verfasst habe.

¹ gemäß § 9 Abs. 1 der Promotionsordnung der TU Darmstadt