Deliberative Systems for Autonomous Robotics: A Brief Comparison Between Action-oriented and Timelines-based Approaches

Pablo Muñoz and María D. R-Moreno
Departamento de Automática, Universidad de Alcalá
Ctra. Madrid-Barcelona Km 33,600 E-28871
Alcalá de Henares, Madrid

Abstract

Autonomous robotics is a multidisciplinary and widely research area. Several approaches have been followed to make autonomous and reliable control architectures, especially for applications in environments inaccessible to humans. We are going to focus on two architectures designed to automate the operation of an exploration robot: on the one hand we have the Model-Based Architecture and on the other hand the Goal-Oriented Autonomous Controller.

On these architectures there is at least one deliberative entity, which is the responsible of managing the long term planning to accomplish the objectives of the mission for which it has been designed for. The first architecture implements an action-oriented planner that uses the Planning Domain Definition Language (PDDL) to describe the environment, available actions and goals. The second system employs a timelines approach, which reasons about temporal state variables using the Domain Definition Language (DDL) to model the physics interactions in the world. In this paper we present the deliberative models that these architectures require to perform a robotic exploration mission, and then, we present a briefly and initial analysis about the differences between the solutions given by both planners, focusing on the flexibility of the deliberative process and the semantics and representative capabilities of the languages employed.

Introduction

Since the birth of modern robotics, important efforts have been made in designing and implementing architectures that allow a robot to autonomously inter-operate with its environment to perform specific tasks. The field in which more research is conducted in autonomous control architectures is exploration robots, whether underwater or space, due to the extreme conditions that make them inaccessible to humans.

The long term planning and scheduling is a primary topic in the design of these autonomous architectures for robotics control. In this way, there are several approaches to solve the deliberative necessities for new science missions such as Curiosity or Mars Science Laboratory (MSL) from NASA or the ESA ExoMars, intended to be launched in 2018. The operation of these robots is a complex task, due to the hard environment conditions, the communications problems (i.e. delays between the ground operation team and the communication windows) and the hard constraints required to the survival of the mission. So, these constraints will require of increasingly capable systems to achieve the goals they are designed for.

In this paper we present a small comparison between the deliberative capabilities of two autonomous architectures designed to automate the operation of a rover-like robot: on the one hand the MoBAr (Model-Based Architecture) (Muñoz, R-Moreno, and Martínez 2011) and, on the other hand, the GOAC (Goal-Oriented Autonomous Controller) system (Ceballos et al. 2011). These architectures employ two distinct approaches to the planning problem as R-Moreno et al. (2008) differentiate:

- Action-oriented: it uses predicates logic and the world is seen as an entity that can be in different states. The domain specifies actions that can be performed to change the state of the world and only applicable when some particular states are set. The objective is to find a sequence of actions that, from an initial world state, through applying successive actions, the system achieves a desired goal state. This approach is followed by the MoBAr architecture deliberative layer.

- Timelines-based: this one is more recent than the action-oriented and it is based on the first order logic. It represents the world in terms of functions that describes the behavior of the system from a time perspective: a timeline is a logical structure used to represent and reason about the evolution of an attribute over a period of time. Rules must be defined to specify how the timelines can change, in order to obtain a sequence of decisions from the planner that bring the set of temporal functions to a final state in which a set of constraints are satisfied. The GOAC deliberator uses this approach.

The paper is structured as follows: first, there is a brief description of these two architectures, focused on the deliberative entity. Next section presents an example problem and the deliberative models that each control architecture uses to solve it. Then we present a brief analysis about the capabilities and lacks of each model. Finally some conclusions are outlined.

Model-Based Architecture

The MoBAr architecture initially developed for the Ptinto robot **??** corresponds to a three layers (3T) system (Gat 1998), in which the top tier will be in charge of the deliberative process, long-term memory and learning process as a function of events that occur in the environment. The middle level or execution system also has a short-term memory, as well as a series of rules that trigger the reactive behavior implemented, in order to response in a short time to eventual situations that may occur in both, the environment and the internal state of the robot. Finally, the low level or functional level, is responsible of providing the functionality of the robot, and relay the information collected by the sensors.

Each layer is based on a model with different levels of abstraction. A model defines the properties, capabilities and constraints of the robot at each layer. Starting from the functional layer, which represents the hardware abstraction level, that is, the definition of the internal state of the robot plus the abilities that it has, the upper layers have fewer detailed models, and thus, less coupled with the underlying hardware. In this way, the executor is in charge of taking high level actions coming from the deliberator and decomposing them into lower level commands supported by the functional layer. So, we want that the executor model has little relationship with the hardware, and the high level model only knows the hardware in terms of goal oriented actions and high level abstraction of constraints.

To implement the MoBAr architecture we have taken advantage of different general purpose technologies: for the deliberative layer we have used the PDDL language (McDermott 1998) and a PDDL-based planner. We have chosen the Universal Executive and its language, PLEXIL (PLan EXecution Interchange Language) (Verma et al. 2006) to model and control the executor, and, finally the G^{en}oM2 (Generator Of Modules) framework (Mallet, Fleury, and Bruyninckx 2002) for the definition and implementation of the functional layer. The first two layers, deliberator and executor, use models represented by a language that will be interpreted by a program, and thus, are easily interchangeable in order to adapt them to multiple robots. Instead, the functional layer is strongly dependent on the hardware.

The architecture follows a sense-(re)plan-act cycle (see fig. 1): at first time, the system takes the initial state of the world (state from the functional layer and the PDDL problem), the goals to achieve and it obtains a feasible plan. This plan is static: the executor takes each action, decompose it in a set of commands and send it to the functional layer. With the results of these commands execution, the executor checks if there is an unexpected situation, and, if it cannot handle that situation, it updates the information of the world, that is goals currently achieved and the actual state of the robot, and then the planner must obtain a new plan.

Focusing on the deliberative capabilities, PDDL-based planners are systems that use two input files to represent their knowledge base. One of the files contains a description of the actions that represent "what can/cannot be done" and the other file includes the three elements which define the problem: the known objects of the world, the initial state and the goals we want to achieve. With this information, the planner searches a sequence of actions that can reach the goals from the initial state. The optimal solution of the problem is a conjunction of two factors: the metric used and the resolution algorithm.

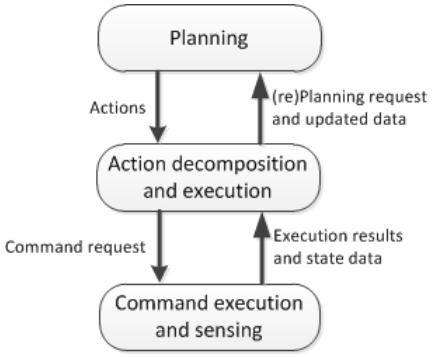

Figure 1: Execution cycle in MoBAr.

Currently we are using the SGplan$_6$ (Hsu and Wah 2008) PDDL-planner that accepts PDDL version 2.1 (Fox and Long 2003) and common features of PDDL 3 (Gerevini and Long 2005). With PDDL 2.1 we can determinate how long each action will take and, using fluents, we can establish a basic resource model for the energy consumption of the actions, and consistently decide whether a plan is feasible or not in terms of the total amount of energy consumed. Also, using version 3, we can employ goals as preferences, so if there is an unreachable goal, we can obtain a plan that ignores it.

Goal-Oriented Autonomous Controller

The GOAC system is the result of a multi-institutional[1] effort within the on-going Autonomous Controller Research and Development activity funded by ESA-ESTEC. GOAC is an architecture that integrates four mature technologies: the functional layer is implemented with the couple G^{en}oM3 (Mallet et al. 2010) plus BIP (Basu, Bozga, and Sifakis 2006) to ensure generation of reliable, modular and verifiable functional components for the hardware abstraction layer; T-REX (McGann et al. 2007; Rajan et al. 2009) is the responsible of the planning and execution dispatching using a timeline-based representation and an interleaving schema (which is similar to IDEA (Aschwanden et al. 2006)); and, finally, the deliberation process resides in the APSI planner (Fratini et al. 2011), a timeline-based, domain independent planner. The deliberative layer of GOAC is composed by a set of one or more deliberative reactors, that is, the couple of a T-REX reactor and an APSI planner. This schema could be seen in fig. 2.

Each deliberative reactor follows a sense-plan-act paradigm for goal oriented autonomy. This set is not fixed, for each robot or mission it could have a different design by defining various reactors and their interactions, giving a scal-

[1]GOAC involves these partners: LAAS-CNRS (France), VERIMAG (France), MBARI (United States of America), CNR-ISTC (Italy) and GMV (Spain).

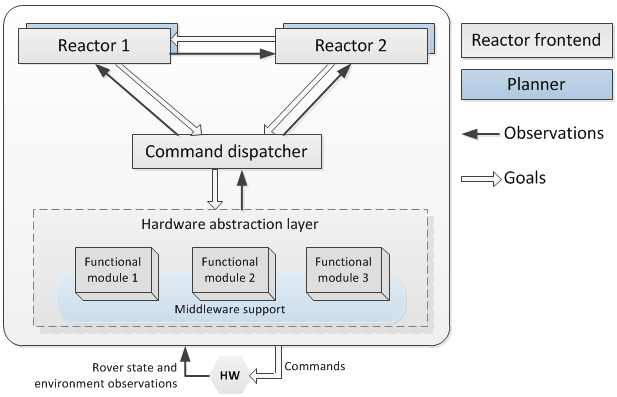

Figure 2: A possible instance of the GOAC architecture.

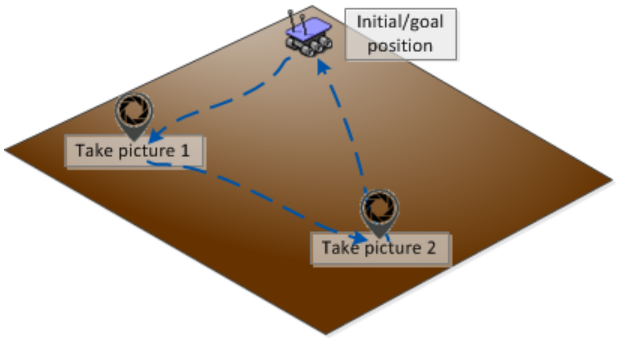

Figure 3: Problem representation.

able architecture. That allows a divide and conquer approach in which the scope of each deliberative reactor (those ones in charge of the planning process using the APSI planner) could be refined by other more specific reactors. To do this, reactors can interact with other sending goals and receiving observations.

In order to ensure the correct execution of the required commands to achieve the goals, it is possible to have one or more reactors in charge of the command dispatching between the deliberative reactors and the functional layer. Considering the fig. 2, there is only one of this kind of reactor, called "command dispatcher". This reactor implements a procedural executive that not only gives correctness in the command execution, it also gives a better level of abstraction and an interface between the functional layer and the higher level reactors. In fact, this reactor is responsible of gathering observation from the functional layer and to provide it in a convenient way to the deliberative reactors.

The deliberative process resides in the APSI planner. For each deliberative reactor there is one instance of an APSI planner, and every one has its own look-ahead window over which to deliberate. APSI uses a timelines representation, which encapsulates an evolution of a particular state variable over time. To define the world it implements the Domain Definition Language (DDL) language (Cesta and Oddi 1996; Fratini, Pecora, and Cesta 2008) that enables to specify the allowed state transitions as well as the causal and temporal relationships between state variables, so, the problem is modelled by identifying a set of relevant features whose temporal evolution needs to be controlled to obtain a desired behavior. Therefore, the result of a deliberative process is a sequence of state transitions for each timeline that achieves a specified condition as a planning goal.

Each deliberative reactor reasons about a specific part of the whole domain, being one the high level reactor, which has the more abstract states, and whenever we go down in the reactor "hierarchy", the reactors are more specific. In this way, the high level reactor puts goals in the timelines of other more specific reactors. Like that, a high level goal state could be decomposed into lower level goals states that must be reached by their respective timelines before the completion of the goal state in the high level. To do this, the reactors have two types of timelines: internal and external. The first ones are the timelines that the deliberative reactor can control directly, the reactor can change its value, but usually the value is dependent on one or more external timelines. These are not controllable from the reactor, but it can produce goal states in that timelines required to reach the goal state of its internal timelines.

Rover example problem

To analyse the deliberative capabilities for the two previous architectures, we define a basic scenario for an exploration rover. The scenario, represented in fig. 3, consists of a rover (such as the Dala[2] robot) that must achieve the acquisition of two pictures in different locations (and possibly with different pointing of the pan-tilt unit), and then, return to the start point. Also, during the traverse it could exist visibility windows in which the rover can transmit the data acquired to a station. The objective is to send all the possible pictures taken during these windows.

To safely operate the rover, there is a little set of constraints that must be included in the models:

- The rover is able to move between two points in space given their coordinates (x, y).
- The rover has two navigation modes: one for flat terrain (using a sick laser range finder) and other for rough terrain (using the stereo-vision cameras). The navigation mode is defined by this property of the terrain, which is continuously updated by a functional module. A change in the terrain triggers a signal that must be trapped by the upper layers to ensure the use of the correct navigation mode.
- The pan-tilt unit can move to reach a desired point, given by their angles (α, β).
- During the acquisition of a picture, the pant-tilt unit must be pointing at the desired site and the rover must stay still.
- When the rover is moving, the pan-tilt unit must be pointing to the front, that is, $(0, 0)$.
- It is possible to transmit pictures while the rover is stopped and exist a visibility window.

[2]http://homepages.laas.fr/matthieu/robots/dala.shtml

```
(:durative-action MoveTo
 :parameters (?r - rover ?p1 ?p2 - loc ?n - navmode)
 :duration (= ?duration (/ (distance_to_move ?p1 ?p2) (speed ?r ?n)) )
 :condition (and (over all (has_locomotion ?r))
            (over all (navigation_mode ?r ?n))
            (over all (platine_pos NavCam plat0_0))
            (at start (position ?r ?p1))
            (at start (>= (energy ?r)(*(power_per_m ?r ?n)(distance_to_move ?p1 ?p2)))) )
 :effect (and (at start (not (position ?r ?p1)))
         (at end (position ?r ?p2))
         (at end (decrease (energy ?r)(*(power_per_m ?r ?n)(distance_to_move ?p1 ?p2)))) )
)
(:durative-action MovePlatine
 :parameters (?r - rover ?c - cam ?p1 ?p2 - platpos)
 :duration (= ?duration (time_move_platine ?p1 ?p2))
 :condition (and (at start (platine_pos ?c ?p1))
                 (at start (>= (energy ?r)(*(platine_energy)(time_move_platine ?p1 ?p2)))) )
 :effect (and (at start (not (platine_pos ?c ?p1)))
              (at end (platine_pos ?c ?p2))
              (at end (decrease (energy ?r)(*(platine_energy)(time_move_platine ?p1 ?p2)))))
)
(:durative-action TakePicture
 :parameters (?r - rover ?p - loc ?c - cam ?a - platpos ?m - mode)
 :duration (= ?duration (time_to_picture ?c ?m))
 :condition (and (over all (camera_mode ?r ?c ?m))
                 (over all (position ?r ?p))
                 (over all (platine_pos ?c ?a))
                 (at start (>= (energy ?r) (camera_energy ?c ?m))) )
 :effect (and (at end (picture ?p ?m ?a))
              (at end (decrease (energy ?r) (camera_energy ?c ?m))) )
)
```

Figure 4: PDDL actions definition for the rover example.

We are interested in reviewing how both models designed for this scenario, one modelled in PDDL and the other in DDL, solve the problem. Taking into consideration real constraints in the scenario, such as partially known (and potentially dynamical) environment, energy requirements and possibility of failures; make that problem a good case of study for a basic comparison between models. Next sections briefly explain the high-level models for the MoBAr and the GOAC architectures respectively.

PDDL model

The PDDL model here presented is an adaptation of the employed in the MoBAr architecture (Muñoz, R-Moreno, and Martínez 2011). The problem and domain files contain the knowledge of the rover and the actions that it can perform. The domain, shown in fig. 4, specifies the actions that the robot perform to change both, its internal state and the environment. The actions representation include the duration and the energy consumption. The energy is treated as a fluent, but it is not the best solution; the energy model can be more effective if it were treated as a resource. The system only maintains the value of the fluent, modifying it during the planning, but the planner does not reason about how to deal with it in a efficient manner. For each action, there is a precondition that specifies the amount of energy required to performing the action, and therefore, there is an effect to decrease the energy level consequently.

The movement action (`MoveTo`) is based on the rover navigation mode and the travel distance. The time to travel is dependent on the speed of the rover in the current navigation mode. One relevant condition prior to move is to check that the pan-tilt unit of the navigation cameras is pointing to the front.

The other actions are related to the picture acquisition: `MovePlatine` and `TakePicture`. First one moves the pan-tilt unit associated to a specific camera from a start position to a desired position. The duration of this action depends on the movement, and the energy required is a fixed value multiply by the required time. To take a picture, we need a camera with a determined mode, orientation of its pan-tilt unit and the rover stopped in the desired location. Time spent and energy consumption of the action is a function of the camera and mode employed.

For the previous domain we need to define the present objects of the world, the initial state (including rover energy consumption and time spent to move between locations) and the goals to achieve. The problem presented in fig. 5 includes all the necessary data to solve the rover example problem explained previously. First, we need to define the possible objects, that is, the relevant locations, cameras and modes, pan-tilt positions and, finally, rover and its navigation modes.

For the initial state, the connections between locations must be set, defined by the distance that separates both points. If we want to define different times for each navi-

gation mode, we need to duplicate that info including the navigation mode involved as a parameter. Next there is the rover data. It contains the initial position, energy, navigation mode and the energy consumption of each subsystem. As specified in the domain model, the energy required for the navigation and camera subsystems depend on the current operation mode. Also, there are the functions that specify the duration of the movement for the pan-tilt unit.

The last element of the problem is the goal(s) definition. It defines the tasks that the rover must perform, such as, go to a desired location, or take an image from a location and with a particular orientation and mode. If the planner supports plan preferences, one or more targets cannot be satisfied by the planner, assuming a penalization for each unsatisfied goal. The goals defined for the rover example are: take two pictures and finish the plan in the initial location. The metric defined specifies the goodness of the solution in terms of the time spent and the unsatisfied goals.

DDL model

The domain and problem described here correspond to a case study scenario for the Dala robot employed in a particular instance of the GOAC architecture. For this example there are two deliberative reactors: one in charge of the mission control (the one that accepts the goals from the user), and the other composed of different timelines to represent the rover subsystems (navigation, camera, platine and terrain to define the navigation mode) at which the other reactor dumps lower level objective states necessary to reach high level goals. The domain model represents the physical interaction between the different state variables, and the problem defines the states at the start time, and a set of desired states that must be accomplished at the end of the execution.

A small fragment of the domain is shown in fig. 6. There are two transitions defined, one for the high level deliberator and the other related to the timeline that represents the evolution of the rover position. Both specify the relationship between the current state of the timeline and the requirements to switch to another state. In the first case, there is the definition for a picture acquisition controlled by the *MissionTimeline* timeline. As required by the problem defined previously, the rover must be in a particular location, determined by its coordinate pair, `(?x1, ?y1)` and the pan-tilt unit must be pointing with a desired pair of angles, `(?pan1, ?tilt1)` using the variables expressed in fig. 6. The picture acquired is stored using the `?file_id1` identifier. To express the relationship and the changes triggered when the *MissionTimeline* starts the execution of this transition, there is the definition of the state required in the other timelines: `cd1` implies that the timeline in charge of the camera must change to acquire a picture in the desired position and pointing; and `cd5` requires from the communication subsystem to transmit that picture to the station. When these two subgoals are achieved by their respective timelines, the high level timeline, *MissionTimeline*, go back to the idle state, as expressed in `cd2`. Obviously, prior to transmit the picture, it must be taken; this relationship is declared using `cd1 BEFORE [0,+INF] cd5`: when `cd1` is completed, in an interval that starts immediately after the conclusion of `cd1`

```
(define (problem RoverDemoProb)
 (:domain RoverDemo)
 (:objects
     C0_0 C6_0 C5_-5 - loc
     NavCam - cam  lowRes - mode
     plat0_0 plat15_30 - platpos
     laser stereo - navmode
     dala - rover
 )
 (:init
  ;LOCATIONS INTERCONNECTION
  (= (distance_to_move C0_0 C6_0) 10)
  (= (distance_to_move C6_0 C0_0) 10)
  (= (distance_to_move C0_0 C5_-5) 8)
  (= (distance_to_move C5_-5 C0_0) 8)
  (= (distance_to_move C5_-5 C6_0) 6.4)
  (= (distance_to_move C6_0 C5_-5) 6.4)
  ;ROVER DATA AND CONFIG
  (position dala C0_0)
  (= (energy dala) 1.44)
  (has_locomotion dala)
  (navigation_mode dala laser)
  (= (speed dala laser) 0.2)
  (= (speed dala stereo) 0.1)
  (= (power_per_m dala laser) 0.002)
  (= (power_per_m dala stereo) 0.034)
  (camera_mode dala NavCam lowRes)
  (= (camera_energy NavCam lowRes) 0.008)
  (= (time_to_picture NavCam lowRes) 15)
  (platine_pos NavCam plat0_0)
  (= (platine_energy) 0.003)
  (= (time_move_plat plat0_0 plat15_30) 10)
  (= (time_move_plat plat15_30 plat0_0) 7)
 )
(:goal (and
 (preference PIC1
   (picture C6_0 lowRes plat15_30))
 (preference PIC2
   (picture C5_-5 lowRes plat15_30))
 (preference POSF (position dala C0_0)) )
)
(:metric minimize (+
 (*(is-violated PIC1) 100)
 (*(is-violated PIC2) 100)
 (*(is-violated POSF) 900)
 (total-time) ))
)
```

Figure 5: PDDL problem for the rover example.

and infinity, `cd5` can occur. The `TakingPicture` state in the *camera* timeline implies also a decomposition of states to other timelines prior to the picture acquisition: first, the rover must be at the desired location, then the pan-tilt unit needs to be pointed to a specific angle and, finally, the picture can be taken.

The other definition presented here is the one that controls the behavior of the rover movement. In this case, is related to the movement over flat surfaces using the laser navigation mode. So, there is a check condition (`cd3`) to ensure that the terrain is flat prior to move using the laser navigation. Also, as specified in the domain definition, the pan-tilt

```
SYNCHRONIZE MissionTimeline.mission_timeline {
 VALUE TakePicture(?file_id1, ?x1, ?y1, ?pan1, ?tilt1) {
   cd1 Camera.camera.TakingPicture(?file_id2, ?x2, ?y2, ?pan2, ?tilt2);
   cd5 Communication.communication.Communicating(?file_id3);
   cd2 MissionTimeline.mission_timeline.Idle();
   CONTAINS [0,+INF] [0,+INF] cd1;
   CONTAINS [0,+INF] [2,2] cd5;
   MEETS cd2;
   cd1 BEFORE [0, +INF] cd5;
   ?x1 = ?x2;
   ?y1 = ?y2;
   ?pan1 = ?pan2;
   ?tilt1 = ?tilt2;
   ?file_id1 = ?file_id2;
   ?file_id1 = ?file_id3;
} }
SYNCHRONIZE RobotBase.robot_base {
 VALUE GoingToLaser(?x1, ?y1) {
   cd2 Platine.platine.PointingAt(?pan1 = 0, ?tilt1 = 0);
   cd3 Terrain.terrain.FlatTerrain();
   DURING [0, +INF] [0, +INF] cd2;
   DURING [0, +INF] [0, +INF] cd3;
} }
```

Figure 6: Fragment of the DDL domain definition.

```
PROBLEM RoverDemoProb (DOMAIN RoverDemo) {
mtl0 <fact> MissionTimeline.mission_timeline.Idle() AT [0,0] [1,+INF][1,+INF];
comvw1 <fact> Communication.communication_windows.Visible() AT [70,70][90,90][20,20];
comvw2 <fact> Communication.communication_windows.Visible() AT [150,150][180,180][30,30];
pos0 <fact> RobotBase.robot_base.At(?x1=0, ?y1=0) AT [0,0] [1,+INF] [1,+INF];
or0 <fact> Platine.platine.PointingAt(?pan1=0, ?tilt1=0) AT [0,0][1,+INF][1,+INF];
cam0 <fact> Camera.camera.CamIdle() AT [0,0][1,+INF][1,+INF];
com0 <fact> Communication.communication.CommIdle() AT [0,0][1,+INF][1,+INF];
tp1 <goal> MissionTimeline.mission_timeline.TakePicture
    (?gfile_id2=1, ?gx2=6, ?gy2=0, ?gpan2=15, ?gtilt2=30) AT [10,10][80,90][70,80];
tp2 <goal> MissionTimeline.mission_timeline.TakePicture
    (?gfile_id1=2, ?gx1=5, ?gy1=-5, ?gpan1=15, ?gtilt1=30) AT [80,95][160,185][80,90];
gp <goal> MissionTimeline.mission_timeline.At(?gx3=0, ?gy3=0) AT[160,190][161,+INF][1,+INF];
}
```

Figure 7: Problem definition for the rover example coded in PDL.

unit must be pointing at $(0, 0)$ in order to proceed to move. This is expressed in the condition `cd3`, and if the observation from the timeline that manages the pan-tilt unit shows that the orientation is not the required one, a transition is injected in that timeline prior to the navigation transition. These two conditions must be satisfied during all the time that the `GoingToLaser` state is active.

The problem definition expressed in the Problem Definition Language (PDL) for the example case we are dealing with is shown[3] in fig. 7. This, as well as the PDDL model, establishes the initial state of the world and the goals. To define the initial state is required to set the state of all the timelines employed. For the high level deliberator there are the *MissionTimeline* and *Communication* timelines. First one starts in an idle state, and the other sets the temporal intervals in which there is a communication window available to transmit the pictures, for the example there are two windows, `comvw1` and `comvw2`. The rest of the time is assumed that there is not visibility with the station. For the other rover subsystems there are four timelines that control the rover position, the camera, pan-tilt unit and the communication module. Camera and communication timelines start in idle state, position of the rover is set to the start location $(0, 0)$ and the pan-tilt unit is oriented to the front. For each initial fact (and goal) there are three time intervals: start, end and duration. Every interval is defined by the minimum and maximum time, that is, the transition must start after the minimum time and before the maximum. This applies to the start and end intervals. The duration specifies the minimal and

[3] Although we present the initial facts of all timelines, these values are taken from the state of the functional modules in the GOAC system, they are presented here only to provide an initial state of the system. To work with GOAC there is only required the goal specification and the evolution of uncontrollable timelines, that is, the ones that the system cannot modify, such as the timeline which specifies the visibility windows.

maximal time that the transition can take.

The goal state is declared as a set of states that must be satisfied in a defined temporal interval and, if necessary, in a particular order. Here are defined three objective states, `tp1` and `tp2` and `gp`. First two define the state of acquiring a picture using the same angles for the pan-tilt unit, but in different locations. The constraints defined are that the picture 1, `tp1`, must be taken and transmitted before the second one, `tp2`, and the goal position, `gp`, must be reached after both pictures are taken. The precedence order is provided by the start interval of each goal.

Brief analysis of deliberative models

We are not interested in discussing the optimality of the solution or the time spent to get it, because both architectures use different approaches, and the unique way to obtain a real comparison is to make a hard test-bench and to monitor all the relevant data: energy consumption, required time, etc. Particularly, we are going to push the focus on the flexibility of the deliberation process, which not only resides in the planning/scheduling algorithm, due to the restrictions added by the language employed. In this way, a flexible deliberator must manage system failures and unpredictable elements in the environment such as obstacles in the path or opportunistic science situations. Also, the possibility of including goals during the plan execution is a desirable competence on those systems.

Using the previous models for the action-oriented planning using PDDL and SGplan planner; and the timelines approach with DDL and T-REX/APSI deliberative reactors, we have obtained the plans presented in figures 8 and 9 respectively.

First, we need to analyze how both planners obtain a solution: SGplan (and PDDL-based planners in general) takes a domain and a problem and performs a planning process that achieves all the goals, and then, it returns the sequence of actions to reach the goal state. In the opposite side, APSI timeline-based planner deliberates over a temporal horizon, that is, it gives a partial plan that grows during its execution. Reasoning about that, allows us to see what happens if there is an unexpected situation, for example, if the rover detects a change in the terrain while moving in laser mode and thus, the navigation mode must change to stereo navigation mode to continue:

- The plan obtained with the PDDL model is fixed, so, if the movement action cannot continue in the nominal way, the executor system must manage the situation. In that case, the only way is updating the data of the problem and requesting the planner a new plan. This implies some modifications: first, the necessity of changing the rover data, that is, the current position and energy. But the problem defines only the relevant locations for the initial problem, so the new location needs to be added to the objects and the connections between locations, as well as the destination of the unsuccessful move. Finally, if there are goals already reached, they must be erased from the problem. But, what happens with the navigation mode? We can change the initial state of the navigation mode, but then, the rest of the navigation is performed using the stereo mode, so, when the functional layer triggers another change in the terrain, this process must be repeated.
- The GOAC deliberative reactors contains a set of timelines that control the state of the different subsystems. The current location is managed by a specific timeline (*RobotBase* in fig. 9) that checks another timeline which contains the state of the terrain (*Terrain* in the figure). This state could be flat or rough. When the rover detects the terrain change, there is an observation that modifies the state of that timeline to rough. Considering that *RobotBase* is `GoingToLaser`, as previously explain, this transition required that the *Terrain* is permanently in flat state. So, the transition currently executing in the *RobotBase* timeline cannot continue. But there is another transition defined in that timeline, `GoingToStereo`, that employs the stereo navigation mode. *RobotBase* changes its state to that one, and continues moving to the desired location. In that case, there is no problem to change between laser and stereo navigation modes, the planner can change it dynamically based on the observation of the state of the timelines, which are continuously updated. Also, if the change in the navigation mode does not exceed the time to reach the destination, there is no necessity to perform other changes in high level timelines.

We found that the plan obtained for the MoBAr architecture is a sequence of actions with a fixed duration. What happens if an action takes more or less time to execute? Using a PDDL model, we cannot deal with a flexible model of time, so the only possibility is to use the worst time case for the actions and to delegate the control of the time to the executor. As an example of this problem, we can mention the visibility windows to transmit the pictures. In PDDL we cannot define a specific time in which the rover can transmit the data acquired because the language does not support this type of constraints. It is possible to use a fluent to control the time but it is not practical: it requires to include more complexity to all actions, expressing the evolution of the time manually (the planner does not manage it as it does with the `duration` variable) and to define the communication windows in the problem in terms of start time and end time. However, our experience with this solution do not give good results, if the communication intervals are smaller, the planner does not provide a solution, although it really exits. In contrast, the timelines have a flexible time behavior, when a state transition finishes another transition waiting for the last one can begin. In the same way, we can define the interval in which there are visibility windows, so the transmission of the pictures can be performed in the correct time.

For the case of the goal injection during the execution, a typical situation for exploration missions is to exploit unexpected science targets. We found the same problem as when the rover gets stuck for the PDDL model: we need to modify the initial state and the goals of the problem and perform a replanning to obtain a new plan that includes the new goal. For the timelines approach, a new goal implies the inclusion of the new state and a dynamically replanning process, propagating new subgoals to the different timelines to reach the

```
  0.001: (MOVE_TO  C0_0 C6_0 LASER) [50.0000]
 50.002: (MOVEPLATINE NAVCAM PLAT0_0 PLAT15_30) [10.0000]
 60.003: (TAKEPICTURE C6_0 NAVCAM PLAT15_30 LOWRES) [15.0000]
 75.004: (MOVEPLATINE NAVCAM PLAT15_30 PLAT0_0) [7.0000]
 82.005: (MOVE_TO C6_0 C5_-5 LASER) [32.0000]
114.006: (MOVEPLATINE NAVCAM PLAT0_0 PLAT15_30) [10.0000]
124.007: (TAKEPICTURE C5_-5 NAVCAM PLAT15_30 LOWRES) [15.0000]
139.008: (MOVEPLATINE NAVCAM PLAT15_30 PLAT0_0) [7.0000]
146.009: (MOVE_TO C5_-5 C0_0 LASER) [36.0000]
```

Figure 8: Solution obtained by SGplan for the rover PDDL model.

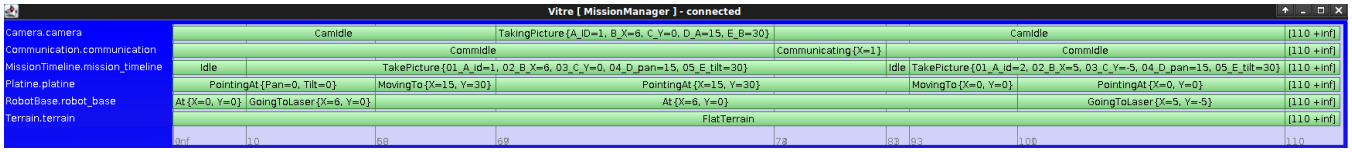

Figure 9: Part of the solution obtained by a particular instance of GOAC using the DDL model previously presented[4].

new objective.

Another difference that appears is the semantic of these approaches: in the case of the PDDL language we need to define all the objects in the environment, what makes it hard to work with continuous values, such as the position of the rover or the orientation angles for the pan-tilt unit. However, DDL is designed to work in continuous domains, defining the interval in which the coordinates or the pan-tilt unit can work, so there is no necessity of defining objects, only set the initial value for the relevant variables. This is more natural than the definition of objects in PDDL, in which we cannot use numerical values.

Finally, an important question in these systems are the resources. For the unitary reusable resources such as a camera or a sample collection tool, both systems could manage the situation in a good manner. In PDDL is easy to define a predicate that indicates when a resource is busy and for the timelines approach, the timeline in charge of the resource expresses when the resource is working or idle, being visible to other timelines that could require it. But for cumulative resources the problem is a more complex question and is not solved yet. Focusing on the rover energy, for the DDL models there is no presence of it, but some work is being carried out in this topic (Diaz et al. 2011; 2012). Meanwhile, the PDDL allows us to define the energy as a fluent, which is possible to consume or renew it within the actions. Model it is not easy and, as is the case of the time, the values are fixed and only allows to represent maximum energy consumption allowed for each action. Also it is possible to maintain the instantaneous power below a threshold using another fluent (here we have not done it). With the PDDL model employed for the battery we can ensure that the plan obtained does not spend more energy than the available.

[4]The image is acquired using the Vitre reactor included in T-REX. This reactor shows the timelines evolution for the deliberative reactors.

Conclusions

This paper has presented the deliberator models for a rover case study for two architectures focused on autonomy: MoBAr and the GOAC architecture, result of the effort of multiple institutions under the ESA supervision.

On the one hand, we have a PDDL-based planner and its corresponding model. The propositional logic is easy to understand and allows us to model basic domains and problems that are action-oriented. The inherent problems are both, the fixed treatment of the time and the static solution provided by the planner. On the other hand, we have deliberators that implement the first order logic in a timelines based approach, which can use the DDL language to model the world. Models are quite complex, but allows us to work with time and continuous variables, giving us more realistic representations of the physics interactions that occur in the environment. Also, dealing with time allows the planning algorithm to deliberate over a dynamic temporal horizon, being more robust to unexpected changes in the nominal conditions of the environment.

So, we can conclude that the deliberative layer of the GOAC system (the couple T-REX/APSI timeline-based planner) give us a better way to model the complex behavior of a robot and its interactions with the environment than the MoBAr deliberator (SGplan a PDDL-based and action-oriented planner) in terms of flexibility of the deliberation process and the semantic employed by their respective modeling languages.

Acknowledgments. Pablo Muñoz is supported by the European Space Agency (ESA) under the Networking and Partnering Initiative (NPI) *Cooperative systems for autonomous exploration missions*. This work was partially supported by the Spanish CDTI project COLSUVH. Authors want to thank Dr. Andrea Orlandini and Dr. Amedeo Cesta for their support with the GOAC architecture.

References

Aschwanden, P.; Baskaran, V.; Bernardini, S.; Fry, C.; R-Moreno, M. D.; Muscettola, N.; Plaunt, C.; Rijsman, D.; and Tompkins, P. 2006. Model-unified planning and execution for distributed autonomous system control. In *AAAI 2006 Fall Symposia*.

Basu, A.; Bozga, M.; and Sifakis, J. 2006. Modeling hterogeneous real-time components in BIP. In *4th IEEE Int. Conference on Software Engineering and Formal Methods*.

Ceballos, A.; Bensalem, S.; Cesta, A.; Silva, L. D.; Fratini, S.; Ingrand, F.; Ocón, J.; Orlandini, A.; Rajan, F. P. K.; Rasconi, R.; and Winnendael, M. V. 2011. A goal-oriented autonomous controller for space exploration. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Cesta, A., and Oddi, A. 1996. *DDL.1: A Formal Description of a Constraint Representation Language for Physical Domains*. Amsterdam: IOS Press.

Diaz, D.; R-Moreno, M. D.; Cesta, A.; Oddi, A.; and Rasconi, R. 2011. Toward a csp-based approach for energy management in rovers. In *4th IEEE International Conference on Space Mission Challenges for information technology (SMC-IT 2011)*.

Diaz, D.; R-Moreno, M. D.; Cesta, A.; Oddi, A.; and Rasconi, R. 2012. An integrated constraint-based, power aware control system for autonomous rover mission operations. In *i-SAIRAS 12 - International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *AI Research* 20:61–124.

Fratini, S.; Cesta, A.; Rasconi, R.; and Benedictis, R. D. 2011. APSI-based deliberation in goal oriented autonomous controllers. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences* 18(2):231–271.

Gat, E. 1998. Three-layer architectures. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Mobile Robots and Artificial Intelligence*, 195–210. AAAI Press.

Gerevini, A., and Long, D. 2005. Plan constraints and preferences in PDDL3. In *The Language of the Fifth International Planning Competition*.

Hsu, C., and Wah, B. 2008. The SGPlan planning system in IPC-6. In *Sixth International Planning Competition*.

Mallet, A.; Pasteur, C.; Herrb, M.; Lemaignan, S.; and Ingrand, F. 2010. GenoM3: Building middleware-independent robotic components. In *2010 IEEE International Conference on Robotics and Automation*.

Mallet, A.; Fleury, S.; and Bruyninckx, H. 2002. A specification of generic robotics software components: future evolutions of GenoM in the orocos context. In *International Conference on Intelligent Robotics and Systems*.

McDermott, D. 1998. The PDDL planning domain definition language. *The AIPS-98 Planning Competition Comitee*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2007. T-REX: A model-based architecture for auv control. In *3rd Workshop on Planning and Plan Execution for Real-World Systems (ICAPS07)*.

Muñoz, P.; R-Moreno, M. D.; and Martínez, A. 2011. A first approach for the autonomy of the exomars rover using a 3-tier architecture. In *ASTRA 2011 - 11th Symposium on Advanced Space Technologies in Robotics and Automation*.

R-Moreno, M. D.; Cesta, A.; and Kurien, J. 2008. Innovative AI technologies for future esa missions. In *ASTRA 2008 - 10th Symposium on Advanced Space Technologies in Robotics and Automation*.

Rajan, K.; Py, F.; McGann, C.; Ryan, J.; Reilly, T. O.; Maughan, T.; and Roman, B. 2009. Onboard adaptive control of AUVs using automated planning and execution. In *International Symposium on Unmanned Untethered Submersible Technology*.

Verma, V.; Jnsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and language for robust spacecraft control and operations. In *American Institute of Aeronautics and Astronautics Space Conference*.