Integrating a PDDL-Based Planner and a PLEXIL-Executor into the Ptinto Robot

Pablo Muñoz[1], María D. R-Moreno[1], and Bonifacio Castaño[2]

[1] Departamento de Automática, Universidad de Alcalá
28805 Alcalá de Henares (Madrid), Spain
{pmunoz,mdolores}@aut.uah.es
[2] Departamento de Matemáticas, Universidad de Alcalá
28805 Alcalá de Henares (Madrid), Spain
bonifacio.castano@uah.es

Abstract. Ptinto is a prototype of a hexapod robot for exploration tasks in rocky and cumbersome areas. The main objective of this prototype is to design and test a complex kinematic control system, including both new hardware and software technologies.

In this paper we describe the autonomous architecture that we have developed for the control of the system. We use a deliberator that must be able to make a safe trajectory between two or more points avoiding obstacles. When a trajectory has been created, the executive takes this plan to control Ptinto via the hardware abstraction layer. This is a classical 3T architecture implementation with two general purpose systems: a PDDL planner as the deliberator, and a PLEXIL executor.

Keywords: Autonomous agents, intelligent system, planning and scheduling, intelligent execution, robotics.

1 Introduction

Since the born of robotics to the present-day there has been a continuous progress in software implementation and new hardware architectures. This progress implies, for instance, that most of the production chains in industry are robotized. In addition to that, there are several robots exploring our Solar System as well. In the last decade we can emphasize some examples such as the NASA Mars Exploration Rovers (MER) [1, 2]. This project was launched in 2003 and, with 90 days of lifetime in Mars, it is currently operative.[1]

In the same way, the software to control these complex hardware systems has made a great progress towards more autonomous and reliable systems. Although exploration robots are tele-operated and have a minimum degree of autonomy, there are several systems as: Intelligent Distributed Execution Architecture (IDEA) [3], Teleo-Reactive EXecutive (T-Rex) [4] or LAAS [5] focused on making full autonomous control architectures.

[1] `http://marsrover.nasa.gov/home/index.html`

Space investigation is a very expensive task, so that, before sending an exploration robot to an unknown environment, it has to be tested thoroughly in similar conditions that are expected at its destination. In Spain there is a location: the Tinto river environment, that has been considered one of the few places on Earth which presents ground conditions very similar to the forecast for the Mars planet: is dense, rich in heavy metals and with limited oxygen. To take advantage of this situation and in order to study that special area, the Astrobiology Centre (CAB-INTA) in Spain is working on a terrestrial exploration hexapod robot called Ptinto. This kind of legged robots offer better mobility on difficult terrains and are more capable to overcome obstacles than the traditional rovers [7]. For example, the MER-A Spirit rover is embedded in soft sand since April 23th of this year and it is not expected to be released until next year.[2] Ptinto has six pods with three degrees of freedom each and three extensiometric gauges bridges per pod. Theses bridges are able to detect obstacles by collision and this info represents the only knowledge the robot can get form its environment so far.

This paper is an extension of our previous work [8] where we developed a basic 3T [6] architecture for the control of the locomotion of Ptinto. This time we presents a more complex autonomous architecture using two general purpose systems: a PDDL planer called SGPlan$_6$ [9] as the deliberator system and the couple PLEXIL language and the Universal Executive [10, 11] as the executor. SGplan$_6$ was the winner of the IPC-8[3] competition under the temporal satisficing track. This has been the main motivation to select it as well as its ability to deal with time, resources and metrics (some of the other planners we were testing could not find any solution to the problems we modelled). PLEXIL has been chosen because it is a general purpose language for execution systems and the UE has a general interface module adaptable to communicate the executor system with our own functional layer.

This new version of the architecture is focused on extending the skills of our system using a more complex execution system. PLEXIL and the UE make easier to incorporate the treatment of time (not present in the previous version), run sequences of commands with their respective monitoring and introduce a fault protection ability. We have also extended the PDDL domain and problem using some features of PDDL3 with variable durations on each action and a new 3-D abstraction of the terrain for the problem and subgoals. The previous version only could manage 2-D terrains and there were not durations in the actions.

The paper is structured as follows. Next section describes the deliberator layer. Section 3 presents the executive language and the associated executor. Then, in section 4 the full architecture of the autonomous control system is described. Finally, we outline some future research lines and the conclusions of our work.

[2] Updates on the efforts to free the Spirit rover in:
`http://www.nasa.gov/mission_pages/mer/freespirit.html`

[3] `http://ipc.informatik.uni-freiburg.de`

2 Deliberator Layer: PDDL Based-Planner

In a typical three tier architecture or 3T there is a top level entity typically called deliberator. The deliberator is a high level system designed to have an "intelligent" behaviour. That implies that the traditional programs are not capable enough to be used under this layer. That is the reason why there have been built domain independent planners to be integrated in these type of architectures.

There are several kinds of planners, some of them are specific and others are generic purpose ones. In this case we have chosen a general purpose planner that uses the Planning Domain Definition Language (PDDL) version 2.1 [12] and common features of PDDL3 [13]. The reason to select a general purpose planner with a standard language is the possibility of replacing the deliberator with a more powerful planner with low cost. The same domain/problem files can be used without modification, and, if the planner support new features of PDDL or extensions (like PIPSS [14], which implements resources as a PDDL extension), we will be able to extend the possibilities of the deliberator system in a short period of time.

In these systems time plays a fundamental role. With PDDL 2.1 we can determinate how long each action will take and establish a maximum duration and consistently decide whether a plan is feasible or not. In addition, the time taken for delivering must be minimized in order to respond in bounded time intervals. The treatment of time has been attended on the architectures mentioned in the introduction: IDEA and T-Rex use a timeline based system. This approach defines both a look-ahead window over which to deliberate and a timeline for each reactive component. In that sense, our system is more similar to the LAAS architecture: the deliberator generates a plan in which every action has a duration and the temporal executor decides when to start or stop these actions. In our case the executor does not modify the sequence of actions, but it executes them concurrently when possible.

PDDL 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 it 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. For the current version of our architecture, we have chosen SGPlan version 6 as the deliberator layer. It is important to mention that SGPlan$_6$ [9] uses parallel decomposition to solve the problem.

3 Executor System: PLEXIL and Universal Executive

An executor is a system that runs between the deliberator and the functional layer. Its purpose is to read the high level orders generated by the deliberator and translate them into a sequence of orders that the functional layer can execute.

However, this is only the first step, because each order in the high level can correspond to one or more in the functional level. At the same time, the executor has to supervise the execution of these orders, because if an error arises during the execution of any of them, this error may provoke that the rest of orders not be executed. In these situations the system will most likely fall into a wrong state or, in the worse case, the entire system will fails. Then, when an error happens the executor must have a response to control that event. In some cases it will try to execute an alternative order, repair the plan or use a redundant system. In other cases it will call back the deliberator with the information about the current state so that it can return a new plan.

For the execution system we have used the couple PLan EXecution Interchange Language (PLEXIL) and its associated executor, the Universal Executive (UE). This project[4] has been developed thanks to the efforts of NASA Ames Research Center, NASA Jet Propulsion Laboratory and Carnegie Mellon University and it has been distributed under the Berkeley Software Distribution license. It has been successfully applied to operate prototype planetary rovers and drilling equipments, and demonstrate automation for the International Space Station operations.

PLEXIL is a structured language to make flexible and reliable command execution plans. It is portable (since it uses XML for the encoding), lightweight, deterministic (given the same sequence of events from the external world) and very expressive. UE is the interpreter, designed to facilitate the inter-operability of execution and planning frameworks. It has an interface module to connect the executor with an external system. This interface can be modified for each particular system.

The UE executes PLEXIL's plans that must have been designed to perform one or more tasks. A PLEXIL plan is a tree of nodes which represents a hierarchical breakdown of tasks. Each node has a set of conditions to control its execution and a section called *body* that describes its function (type of node). With all these elements we can express if-then branches, loops and batch processes and use subplans that would be executed only when a special situation occurs. For instance, there is a plan that manage the fault protection system.

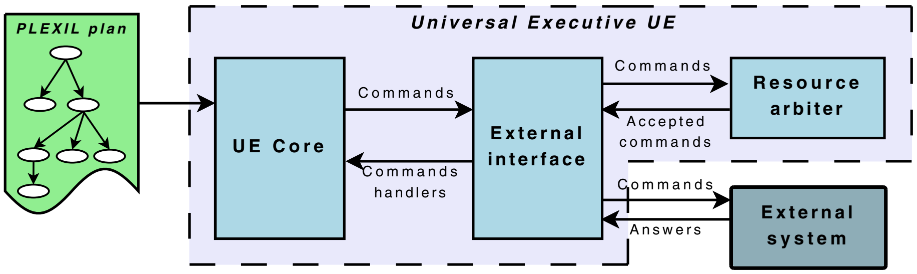


Fig. 1. PLEXIL and the UE

[4] http://plexil.sourceforge.net

The UE executor can be divided into three modules (see Fig. 1). First, the UE core that implements the PLEXIL syntax and the algorithms to execute the plans. Then, there is a module for the management of the resources in case there were conflicts when they were used. Finally, an external interface system to connect the executor with an external system. The UE core algorithm processes each node when it reaches its depth in the tree structure and its starting conditions are fulfilled. When an internal or an external state changes, this change is propagated in cascade until there are no more nodes that can be executed.

4 Control System Architecture

To deal with the Ptinto locomotion control we have implemented a typical 3T architecture. As we have described in the previous sections, the deliberator is a PDDL-like planner (SGPlan$_6$) and the executor is a PLEXIL interpreter (the UE). The functional layer is a C++ specific module for the Ptinto hardware.

These elements are assembled within a main control program written in C++ to interconnect and coordinate all the modules. The flow of the system is expressed below:

1. The control system invokes the PDDL planner with the problem (that contains the model of the ground and the goals to achieve) and the domain files (with the actions that the robot can perform) as inputs. The problem can be manually generated either by a human operator or automatically by a specific system (i.e. an AI vision system). As a result the planner generates a viable plan to reach the objectives avoiding the obstacles.
2. Once the plan has been produced, the executor reads the actions, one by one, and executes them. Each action of the planner can be decomposed in one or more low level commands.
3. Commands are sent from the executor to the functional layer through the executor interface and they are permanently monitored to check whether they are properly executed or not.
4. The functional layer is directly connected to the hardware of the robot and carries out each command issued by the executor. Once the order has been accomplished, the functional layer sends a feedback response to the executor with the outcome of the command.
5. If a command is not properly executed, the executor will try to solve the situation by calling the deliberative layer to replan again. In case this strategy cannot solve the problem, the system would wait for human intervention.

Figure 2 shows the functional view of the architecture, the data and actions involved in each layer and the inputs and outputs that interconnect them. Next subsections explain each one in detail.

4.1 High Tier: PDDL-Like Planner

The high tier is the module in charge of both: generating a plan either when the goals are set for the first time or when the initial plan cannot be reached,

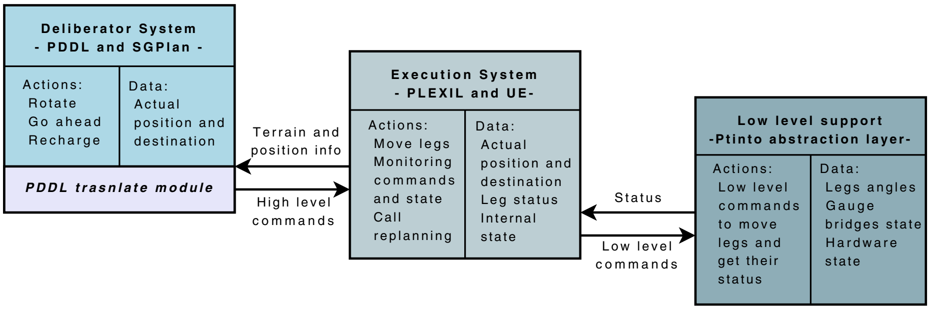


Fig. 2. Functional view of the control system architecture

and updating the problem information. The plan generation is based on the domain and problem definition. The domain definition contains all the actions the robot can perform. This information is static and must be set before Ptinto runs in autonomous mode. The problem definition represents all the information about the world: terrain data (a grid of squares that have aridity and altitude), obstacles, recharge points and data about the state of Ptinto: position, orientation and remaining energy. This information can dynamically change along the time so it is important to keep it updated. The left part of the Fig. 3 shows a conceptual view of a possible scenario. Ptinto is represented as a gray rectangle takin up one square. Each square is defined by its height and degree of aridity. The grid of squares allows us to place elements in the different squares without using a global position coordinates system that would be very complex to be modelled in PDDL. Elements that can be placed in the squares are: obstacles that Ptinto cannot overpass (red elements in the figure), energy recharge points (a blue and white square), points that we want to visit and the goal position (the green arrow). The right part of the Fig. 3 shows also a small portion of the domain and the problem in the PDDL syntax.

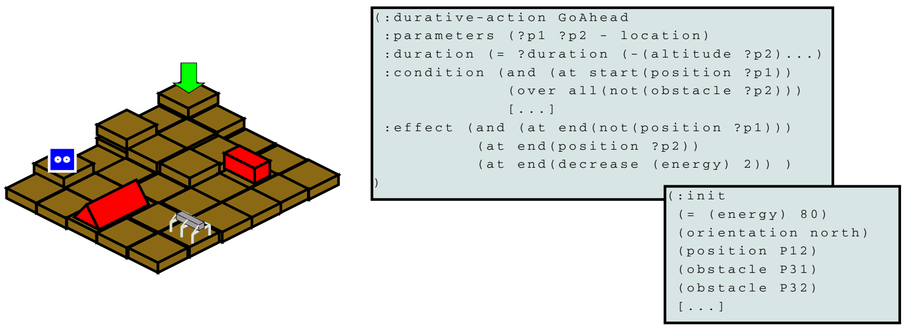
The actions defined in the domain for Ptinto are as follows:

– *Rotate*: changes the orientation of the robot. The possible orientations are: north, south, east and west. The duration of this action is calculated as shown in Eq. 1 where *angle* is the rotation angle in degrees.

$$Duration = \frac{3 \cdot angle}{90} \quad . \tag{1}$$

– *Go ahead*: moves the robot one square in the current direction. This action implies both the difference in height between squares is less than a previously established value and there is no obstacle in the destination square. The cost of this action is a function of the difference in altitude between squares and the aridity of terrain. The right part of Fig. 3 shows how this action is coded in PDDL. The duration for this actions is shown in Eq. 2 where *C1* and *C2* are the origin and destination squares respectively.

$$Duration = (C2\,altitude - C1\,altitude)^2 + C1\,aridity + C2\,aridity + 1 \quad . \tag{2}$$



```
(:durative-action GoAhead
 :parameters (?p1 ?p2 - location)
 :duration (= ?duration (-(altitude ?p2)...)
 :condition (and (at start(position ?p1))
                 (over all(not(obstacle ?p2)))
                 [...]
 :effect (and (at end(not(position ?p1)))
              (at end(position ?p2))
              (at end(decrease (energy) 2)) )
)
```

```
(:init
 (= (energy) 80)
 (orientation north)
 (position P12)
 (obstacle P31)
 (obstacle P32)
 [...]
```

Fig. 3. Graphical representation of how the world has been modelled

– *Recharge*: this action can only be performed if the square has a recharge point. Recharge implies either waiting for a battery recharge or substitution by a human action. How long this action takes is manually added to the problem file.

The output generated by this tier is a sequence of actions that defines the movements that Ptinto has to carry out to accomplish the path between the initial and the goal points. This trajectory will avoid the obstacles and will minimize the energy or the total plan time.

4.2 Middle Tier: PLEXIL Executor

The intelligent executor, or middle tier, is responsible for: reading the high level orders, transforming them into a sequence of low level orders and safely executing them in the correct order. The last part implies that the executor should be monitoring closely whether all the commands are accurately accomplished.

PLEXIL works as follow: first of all we model a PLEXIL plan (a tree of nodes that represent a hierarchy breakdown of tasks) and send that plan to the UE which interprets and executes it. Then, we have to implement the external interface of the UE to link the executor and the functional layer together in order to send commands and get the answers. The connection between the executor and the planner is different. Since the executor cannot read the output of the planner, this information comes through the PDDL translate module. Also, when the executor needs to modify the PDDL problem file (to include both the new ground information and the current position and orientation of the robot), it sends that information to the PDDL translate module and it modifies the archive with the problem accordingly.

The modelled plan should control the moves of the legs, maintain the current position of the robot, and keep the information about the terrain. To accomplish these tasks, the PLEXIL plan is divided in four interconnected plans as seen in Fig. 4:

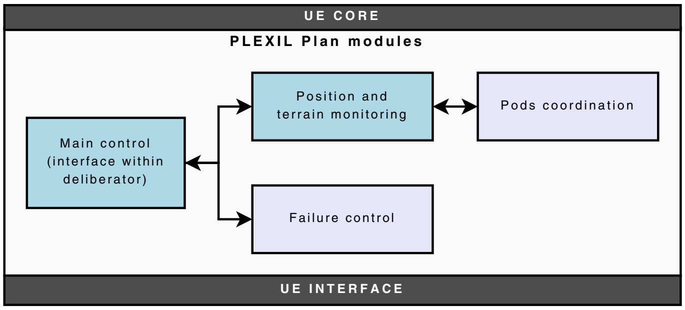

UE CORE
PLEXIL Plan modules
Main control (interface within deliberator)
Position and terrain monitoring
Pods coordination
Failure control
UE INTERFACE

Fig. 4. Modules of PLEXIL plan

- *Main control*: This is the general plan that controls the other subplans. Its function is to read and interpret the high level actions inside the plan generated by the high tier. The necessary information to execute a movement is made up of: the action type (move, rotate or recharge), the origin and destination positions (for moves only), the current and new orientation (for rotations), and how long each action last. With this information the plan controls the execution of the subplans and, when necessary, it calls the high layer to get a new plan according to the new information collected from the terrain and the current position and orientation of the robot.
- *Position and terrain monitoring*: this plan has three functions. (1) To monitor the position of the robot while it is moving. (2) To calculate the abrupt degree of the terrain from the information collected by the gauges and the angle of each pod. This degree sets how safely is going to be the move. (3) When a move cannot be performed, the plan uses some rules to try a new move or determine that there is an obstacle that cut off the present route.
- *Pods coordination*: the coordination of the pods provides the legal moves available for each leg when the robot have to, for instance, make a turn or go forward. One action invoked by this plan is translated into a low level set of commands that will be accepted by the functional layer. This plan is modelled to make concurrent moves when possible. Also, this module receives the results of the executed orders while it is monitoring the correct execution of the movements.
- *Failure control*: when an unknown failure is detected, this plan tries to determine what has failed and try to solve the problem. If there is not routine able to handle the situation, this subplan activates the operator mode and waits for human intervention.

4.3 Low Tier: Ptinto Hardware Control

The Ptinto low tier provides a simple and high level access to the hardware components such as the lineal actuators, the CAN bus or the extensiometric gauges. It also implements some features that guarantee that the access to the hardware is made in safe conditions. This is done with the typical programming techniques such as semaphores and correct synchronization of threads. For example, it checks if two or more move commands are sent to one linear actuator at the same time, or write a new message in the bus when there is another that has not been read.

Another functionality is to provide the commands execution results. This information represents the current state of each leg. That is, the angle of the three lineal actuators and the value of the three extensiometric gauges.

To test the whole architecture we are developing a full environment simulator. At present, the simulator is a basic program that sends the answers through an emulated bus with the angle and gauge values associated to each pod.

5 Conclusions and Future Work

This paper has described the autonomous architecture for the control of the Ptinto robot. The architecture integrates third party software, such as SGPlan$_6$ for the deliberative layer and the PLEXIL system for the executor, into a system that solves a concrete problem: the movility of Ptinto.

The purpose of this work is to integrate a 3T architecture with the possibility to extend the funcionality of the system, if some few changes are needed, and apply it to Ptinto. This means that replacing the PDDL-based planner in the deliberative layer or the PLEXIL executor with a different one, could be done without difficulty. As a consequence, the developed architecture would be fully reusable, and it could be easily adapted to a new layout with lower cost and less time effort than it will take building a new one from scratch.

It is important to mention that some of the decisions performed in the control architecture have been conditioned by the lack of visual sensors in the robot. That is the reason why we have designed a modular architecture that easily will allows us to introduce new developments when any improvement was set up in Ptinto layout. For instance, a vision camera or an ultrasound sensor, will allow us to build an artificial vision system that can automatically generate the terrain information where Ptinto should move. So that, our future work is related to the new hardware that may be included into Ptinto.

Acknowledgements

The authors want to thank Javier Gomez de Elvira, Julio J. Romeral-Planelló and Sara Navarro-López from the CAB-INTA for their support with the Ptinto robot. This work is funded by the Castilla-La Mancha project PEII09-0266-6640.

References

[1] Bresina, J., Jonsson, A., Morris, P., Rajan, K.: Activity Planning for the Mars Exploration Rovers. In: Procs. of the Workshop on Scheduling and Planning Applications of the International Conference on Automated Planning and Scheduling (ICAPS), Monterey, CA, USA (2005)

[2] Bresina, J., Morris, P.: Mission Operations Planning: Beyond MAPGEN. In: Procs. of the Second IEEE International Conference On Space Mission Challenges for Information Technology, Pasadena, CA, USA (2006)
[3] Aschwanden, P., Baskaran, V., Bernardini, S., Fry, C., R-Moreno, M.D., Muscettola, N., Plaunt, C., Rijsman, D., Tompkins, P.: Model-Unified Planning and Execution for Distributed Autonomous System Control. In: Procs. of the AAAI 2006 Fall Symposia. Washington, DC, USA (October 2006)
[4] McGann, C., Py, F., Rajan, K., Thomas, H., Henthorn, R., McEwen, R.: A Deliberative Architecture for AUV Control. In: Procs. of the Conference on Robotics and Automation, Pasadena, CA, USA (2008)
[5] Ingrand, F., Lacroix, S., Lemai-Chenevier, S., Py, F.: Decisional Autonomy of Planetary Rovers. Journal of Field Robotics 24(7), 559–580 (2007)
[6] Gat, E.: Three-Layer Architectures. In: Kortenkamp, D., Bonasso, R.P., Murphy, R. (eds.) Mobile Robots and Artificial Intelligence, pp. 195–210. AAAI Press, Menlo Park (1998)
[7] Smith, T.B., Barreiro, J., Chavez-Clemente, D., Smith, D.E., SunSpiral, V.: ATHLETEs Feet: Multi-Resolution Planning for a Hexapod Robot. In: Procs. of the Workshop on Scheduling and Planning Applications of the International Conference on Automated Planning and Scheduling (ICAPS), Sydney, Australia (September 2008)
[8] Muñoz, P., R-Moreno, M.D., Gómez-de-Elvira, J., Navarro, S., Romeral, J.: An Autonomous System for the Locomotion of a Hexapod Exploration Robot. In: Procs. of the third IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT 2009), Pasadena, CA, USA (July 2009)
[9] Hsu, C.W., Wah, B.W.: The SGPlan Planning System in IPC-6. In: Sixth International Planning Competition, Sydney, Australia (Sepember 2008)
[10] Universities Space Research Association (USRA). PLEXIL and Universal Executive Reference Manual, Tech. Rep. 0.90, NASA (June 2009)
[11] Verma, V., Jónsson, A., Pasareanu, C., Iatauro, M.: Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations. In: Procs. of the American Institute of Aeronautics and Astronautics Space 2006 Conference, San Jose, CA, USA (September 2006)
[12] Fox, M., Long, D.: PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of AI Research 20, 61–124 (2003)
[13] Gerevini, A., Long, D.: Plan Constraints and Preferences in PDDL3. Tech. Rep., Dept. of Electronics for Automation, University of Brescia, Italy (August 2005)
[14] Plaza, J., R-Moreno, M.D., Castaño, B., Carbajo, M., Moreno, A.: PIPSS: Parallel Integrated Planning and Scheduling System. In: Procs. of the 27th Annual Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2008), Edinburgh, UK (December 2008)