Reliable Control Architecture with PLEXIL and ROS for Autonomous Wheeled Robots

Héctor Cadavid[1], Alexander Pérez[1], and Camilo Rocha[2](✉)

[1] Escuela Colombiana de Ingeniería Julio Garavito, AK 45 No 205-59, Bogotá, D.C., Colombia
`{hector.cadavid,alexander.perez}@escuelaing.edu.co`

[2] Pontificia Universidad Javeriana, Calle 18 No 118-250, Santiago de Cali, Colombia
`camilo.rocha@javerianacali.edu.co`

Abstract. Today's autonomous robots are being used for complex tasks, including space exploration, military applications, and precision agriculture. As the complexity of control architectures increases, reliability of autonomous robots becomes more challenging to guarantee. This paper presents a hybrid control architecture, based on the *Plan Execution Interchange Language* (PLEXIL), for autonomy of wheeled robots running the *Robot Operating System* (ROS). PLEXIL is a synchronous reactive language developed by NASA for mission critical robotic systems, while ROS is one of the most popular frameworks for robotic middleware development. Given the safety-critical nature of spacecraft operations, PLEXIL operational semantics has been mathematically defined, and formal techniques and tools have been developed to automatically analyze plans written in this language. The hybrid control architecture proposed in this paper is showcased in a path tracking scenario using the Husky robot platform via a Gazebo simulation. Thanks to the architecture presented in this paper, all formal analysis techniques and tools currently available to PLEXIL are now available to build reliable plans for ROS-enabled wheeled robots.

Keywords: Robot autonomy · Plan Execution Interchange Language (PLEXIL) · Robot Operating System (ROS) · Control architectures · Formal verification · Rewriting logic · Automatic reachability analysis

1 Introduction

Wheeled robots are popular because of the simplicity and versatility of their components. Nowadays, autonomous wheeled robots are being used for complex tasks, including space exploration, mission-critical military applications, and precision agriculture. The integration of a vast amount of sensors and their interaction with the environment, make these robots highly concurrent. This

Camilo Rocha—The first two authors have been supported in part by grant DII/C008/2016 funded by Escuela Colombiana de Ingeniería.

scenario poses a significant challenge for guaranteeing correct behavior such as safe navigation, precise goal tracking, and fault tolerance [2]. Moreover, because of the high value of some of the components and the costs associated to malfunction (e.g., money investment, human lives, or crop production), autonomous robots are expected to be highly reliable [13].

The robotics community has developed near-real simulation environments to test control architectures in operation conditions that can help in design-testing a robot development before its deployment. Nevertheless, due to the intrinsic limitations of simulation-based testing, these environments are far from answering the important question of, up to a high degree of confidence, how reliable control architectures really are. The inherent non-deterministic nature of robot autonomy, and the divide between the mathematical properties and the operational semantics of the language used to program the robot, can make this situation more dramatic. For example, an extensive test-guided validation in the design phase of a robot development could suggest absence of deadlocks in a control architecture. However, deadlocks may be present in the resulting implementation because of semantic issues with synchronization structures in the programming language.

Today's formal methods are scaling up to meet the challenges in the development of mission-critical software and hardware. The notion of software/hardware verification via automatic reachability analysis or model checking has evolved to become an accepted technology in development processes. Moreover, new symbolic techniques based, e.g., on the satisfiability modulo theories (SMT) approach are being readied for industrial use as a solution to the state-explosion problem, commonly faced in the algorithmic verification of concurrent systems. Nowadays, formal methods are being used in the development, e.g., of unmanned aircraft systems [15] and have had big impact in the design of policies for unmanned aircraft systems [25]. A key benefit of the formal methods approach is that it provides techniques, tools, and insights highly valuable in a mission-critical development.

This paper presents a hybrid control architecture, based on the *Plan Execution Interchange Language* [6] (PLEXIL), for autonomy of wheeled robots running the *Robot Operating System* [8] (ROS). PLEXIL is a synchronous reactive language created by NASA for mission-critical robotic systems. ROS is one of the most popular frameworks for robotics middle-ware development. The proposed hybrid architecture is given in a deliberative/reactive/driver layered setting. PLEXIL is used in the reactive layer to implement behavior in a mission plan with a hierarchical composition of nodes, where the ones at the top represent high-level behavior and the ones at the bottom basic actions (e.g., primitive robot commands and variable assignment). On the other hand, ROS (and its drivers) is used to represent components specific to wheeled robots as a hardware abstraction at the level of the robot's sensors. The deliberative layer, where mission plans are generated according to some given goals, is not directly addressed in this paper. The technical contribution of this paper can now be better explained. The integration between PLEXIL and ROS is a bi-directional software adapter between the reactive and driver layers. This adapter enables: (i) the execution of PLEXIL low-level commands in a wheeled robot running ROS, and (ii) the interaction of PLEXIL with the external environment via events triggered by sensors under the control of ROS in the driver layer. This approach is showcased in a path tracking scenario using the Husky robot platform via a Gazebo simulation.

The integration of PLEXIL and ROS presented in this paper has the additional benefit of making available to ROS-enabled robots all the formal analysis and verification tools already developed for PLEXIL. These tools include a rewriting logic semantics in Maude [5], an interactive environment for automatic reachability analysis and LTL model checking [22], and automatic symbolic reachability analysis based on rewriting modulo SMT [23]. This integration opens the door of formal verification to a wide class of ROS modules, thus making autonomous planning more reliable. Furthermore, new plans can now be developed following a more rigorous formal methods-oriented approach.

The rest of the paper is organized as follows. Section 2 reviews some related work. Sections 3 and 4 present, respectively, a high-level description of PLEXIL and ROS. Section 5 proposes the layered architecture based on the PLEXIL-ROS integration and Sect. 6 exhibits a proof of concept. Section 7 concludes the paper.

2 Related Work

There is a vast amount of research in the field of control architecture for autonomous robots. Control architectures can be classified in several categories: by the type of interaction between control modules (e.g., hierarchical or centralized architectures), by the type of functionality assigned to each module (e.g., general or specific purpose), and by the way modules interact with the external environment (e.g., event-based or procedural behavior). The latter category, which is closest to the proposal in this paper, can be further classified as deliberative, reactive, or hybrid. In a *deliberative* control architecture, a plan is generated based on a goal and a static model of the environment targeted after successful operation. In a *reactive* control architecture, control commands are generated during operation based on interaction with the environment. In a *hybrid* control architecture, a deliberative agent statically generates plans that will be executed by reacting to the external environment during operation. Fairly complete surveys of works in each category and subcategory are [13,17,27].

There are significant efforts to have reliable and modular hybrid control architectures for autonomy in robotic systems. On the one hand, the main focus is on using languages with mathematically proven properties, e.g., PLEXIL as the intermediate layer for plan representation and execution. On the other hand, the main focus is on defining control structures on top of ROS-enabled systems. Muñoz et al. [16] propose a control architecture for the Ptinto robot, a hexapod robot for exploration in difficult terrains. They use PLEXIL as the intermediate plan specification language and reactive layer for SGPlan (a deliberative planner that automatically partitions large planing problems into subproblems with specific goals). However, in their work, the hardware abstraction layer is proprietary and therefore the architecture is tied to the specific target robot. Jenson et al. [10], propose a control architecture for AMIGO and other ROS-enabled robots to perform tasks in human environments. Their system uses a *hierarchical ordered planner*, a special type of deliberative system in which a predefined set of actions is hierarchically arranged once a goal is set for the autonomous robot. Benjamin et al. [1] propose ROSoClingo, a control architecture with a reactive layer for ROS-enabled robots. Their approach uses *answer set programming* (ASP), a declarative programming paradigm intended to solve NP-hard combinatorial search problems. The proposed architecture encodes adaptive behaviors directly in a declarative knowledge formalism, which requires the addition of reactive capabilities not necessarily available from the target robot.

It is fair to say that the main difference between the above-mentioned works and the proposal in this paper, is that the latter aims at combining the best elements of both worlds: a robust, mathematically verifiable high-level language such as PLEXIL for the reactive layer, and ROS, the actual de-facto standard for robotic middle-ware. This unique combination brings an important advantage to future projects in robotics. Namely, the possibility of having verified – and eventually certified – control architecture software for autonomous robots that use conventional and affordable hardware.

3 The Plan Execution Interchange Language

The Plan Execution Interchange Language [6] (PLEXIL) is a synchronous reactive language developed by NASA to support autonomous spacecraft operations. It has been used on applications such as robotic rovers, a prototype of a Mars drill, and to demonstrate automation capabilities for potential future use on the International Space Station. Programs in PLEXIL, called plans, specify actions to be executed by an autonomous system as part of normal spacecraft operations or as reactions to changes in the environment. The computer system on board the spacecraft that executes plans is called the Universal Executive [26].

3.1 PLEXIL in a Nutshell

A PLEXIL plan consists of a set of nodes representing a hierarchical decomposition of tasks. A leaf node in the tree represents a primitive task such as variable assignment or a command execution, whereas an intermediate node defines the control structure of its descendants such as sequential or concurrent execution. Each node is equipped with a set of conditions that trigger its execution, e.g., a start condition and an end condition. At any time, each node offers information about its execution state: *inactive*, *waiting*, *executing*, *iterationended*, *failing*, *finishing*, or *finished*. There is also information about the termination status of a task: *success*, *skipped*, or *failure*.

When events are reported by interaction with the external environment (e.g., by sensors or timers), the nodes triggered by such events are executed concurrently, updating e.g., local variables, until quiescence. The internal execution of each node, in turn, can trigger the execution of other nodes. Although more than one event can become enabled simultaneously, all parallel operations in PLEXIL are synchronized and will not arbitrarily interleave. This is because PLEXIL semantics is designed under the synchronous hypothesis [18]. One important feature of PLEXIL semantics is that it guarantees determinism (in the absence of external events), which is a convenient property for autonomous programming because it helps in having a clean mathematical semantics.

As an example, consider the simple PLEXIL plan in Fig. 1. It consists of a tree with three nodes: the root node `SamplePlan`, and the leaf nodes `ActionOne` and `ActionTwo`. In this plan, there are two integer variables, namely, `x` and `y`, which are accessible from any node in the plan. The run-to-completion semantics of the leaf nodes depends on the value of the variable `sensorOne`, which is under control of the external environment. For instance, if the value of `sensorOne` becomes 201, then both leaf nodes will execute in parallel. In an asynchronous setting, such an execution would result in an unpredictable outcome because of the race condition in the assignment of `x` and `y`. However, thanks to its synchronous semantics, PLEXIL guarantees a consistent variable swap in this case so that `x` is assigned the value 20 and `y` is assigned the value 10, without any race condition.

```
SamplePlan:{
  Concurrence{
    Integer x=10,y=20;
    ActionOne{
      start Lookup (sensorOne>100)
      x=y;
    }
    ActionTwo{
      start Lookup(sensorOne>200)
      y=x;
    }
  }
}
```

Fig. 1. A very simple PLEXIL plan.

3.2 Rewriting Logic-Based Automatic Analysis

PLEXIL has been designed with verification and validation in mind, and has motivated the development of an important amount of research and tools in the rewriting logic community. Rewriting logic [14] is a semantic framework that unifies a wide range of models of concurrency. Specifications in rewriting logic are called rewrite theories and can be executed in the rewriting logic implementation Maude [4]. By being executable, they benefit from a set of formal analysis tools available to Maude, such as state-space exploration and automata-based LTL model checking. A *rewrite theory* is a tuple $\mathcal{R} = (\Sigma, E \uplus B, R)$ with: (i) $(\Sigma, E \uplus B)$ an order-sorted equational theory with signature Σ, E a set of equations over the set T_Σ of Σ-terms, and B a set of structural axioms – disjoint from the set of equations E – over T_Σ for which there exists a finitary matching algorithm (e.g., associativity, commutativity, and identity, or combinations of them); and (ii) R a finite set of rewrite rules over T_Σ. Intuitively, $\mathcal{R}$ specifies a concurrent system whose states are elements of the set $T_{\Sigma/E \uplus B}$ of Σ-terms modulo $E \uplus B$ and whose concurrent transitions are axiomatized by the rewrite rules R. In particular, for $t, t' \in T_\Sigma$ representing states of the concurrent system described by $\mathcal{R}$, a transition from t to t' is captured by a formula of the form $[t]_{E \uplus B} \rightarrow_{\mathcal{R}} [t']_{E \uplus B}$; the symbol $\rightarrow_{\mathcal{R}}$ denotes the binary rewrite relation induced by R over $T_{\Sigma/E \uplus B}$.

The *ground* rewriting logic semantics of PLEXIL [5] is a rewrite theory $\mathcal{R}_{\mathrm{PLEXIL}}$ with *topsort* $\mathfrak{s}$, meaning that concurrent transitions in the system are mathematically captured by $\rightarrow_{\mathcal{R}_{\mathrm{PLEXIL}}}$ and are over the set $T_{\Sigma,\mathfrak{s}}$ of Σ-terms of sort $\mathfrak{s}$. The *symbolic* rewriting logic semantics of PLEXIL [21,23], based on the rewriting modulo SMT technique, is a rewrite theory $\mathcal{S}_{\mathrm{PLEXIL}}$ with topsort $\mathfrak{s} \times \Gamma$, meaning that symbolic concurrent transitions in the system are mathematically captured by $\rightarrow_{\mathcal{S}_{\mathrm{PLEXIL}}}$ and are over state pairs of the form $(t\,;\varphi)$ with $t \in T_\Sigma(X)_{\mathfrak{s}}$ and $\varphi(X)$ a quantifier-free first-order logic formula under the control of the SMT solver. Intuitively, a symbolic state $(t\,;\varphi)$ in $\mathcal{S}_{\mathrm{PLEXIL}}$ can represent infinitely many concrete states, namely, those states $t\sigma$ for each ground substitution σ satisfying φ. The constraint φ in a symbolic state is used to model the behavior of variables under control of the external environment. Techniques and tools for reachability analysis and LTL model checking with $\rightarrow_{\mathcal{R}_{\mathrm{PLEXIL}}}$ and $\rightarrow_{\mathcal{S}_{\mathrm{PLEXIL}}}$ have been developed. They have been used for detecting the violation of safety properties such as invariants, race conditions, and deadlock freedom. For details, the reader is referred to [22,24].

4 The Robot Operating System

The creation and development of robots involves the interaction and collaboration of several areas of knowledge such as mechanics, electronics, and computer science. For example, once a robotic device has been mechanically designed and its electronic components able to read data from the environment, software artifacts may be developed to pursue autonomy. In general, a large-span robotics project can require a vast amount of collaborative effort – both in time and money. Thus, it would be highly convenient to reuse as many artifacts as possible across similar projects.

An open source initiative promoted by Willow Garage has emerged and, as a result, the Open Source Robotic Foundation (OSRF) has been established recently. The *Robot Operating System* (ROS) has been created by the OSRF with the goal of increasing the reusability of the software components specifically developed for robots. The adoption of ROS can also dramatically decrease the time and money needed to deploy robot applications. During the past few years, ROS has gradually become the de-facto standard in robot development. For example, the paper [19] introducing ROS has been cited 3350+ times since 2009. ROS has not only gained a distinguished position in the research community, but it has also become an important player in the robot manufacturing industry: the ROS Industrial Consortium has the support of 42 of the most prestigious robot manufacturers in the world.

The Robot Operating System is defined in [8] as follows:

> ROS is a meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation 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.

ROS has been designed to be modular at a fine-grained scale, where the basic unit is a *node* representing a process. For example, one node can control a laser range-finder, or the wheel motors, or perform localization. A robot application in ROS can be specified as a computation graph representing the peer-to-peer network resulting from node interaction: an edge in this graph denotes message-passing communication between two processes. Figure 2 depicts a graph corresponding to a ROS program.

Tools are provided by ROS to analyze and visualize data, simulate robots and environments, and to store large amounts of data generated by sensors and processes. For instance, RViz [9] and Gazebo [7] are useful tools to visualize and analyze the data captured by sensors and kinematics. RViz has been designed to visually interact with almost all processes running in a ROS graph. Gazebo, on the other hand, is an open source 3D dynamic simulator [11], which will be included by the OSFR in modern distributions of ROS.

Figure 3 depicts the interaction between the Gazebo simulation environment and a Husky robot deployed in an environment with obstacles. On the left, a computer simulation of the environment can be seen. On the right, a RViz application is shown with the information retrieved by several sensors. For example, data captured from the camera and laser mounted on the Husky has been interpreted and drawn by RViz.

One key effort in ROS has been the standardization of message-passing and the abstraction of the format they use. Messages are important in ROS because they are at the heart of interaction infrastructure. Each message in ROS is a data structure comprising different types of fields such as integers, floating point values, Booleans, and arrays of primitive types. For example, the Twist message format is part of the geometry package in ROS, and is used to communicate linear and angular velocities of a body:

```
Vector3  linear
Vector3  angular
```

The message standardization in ROS makes it possible to create a *clean* hardware abstraction layer (HAL) between hardware and software. In particular,

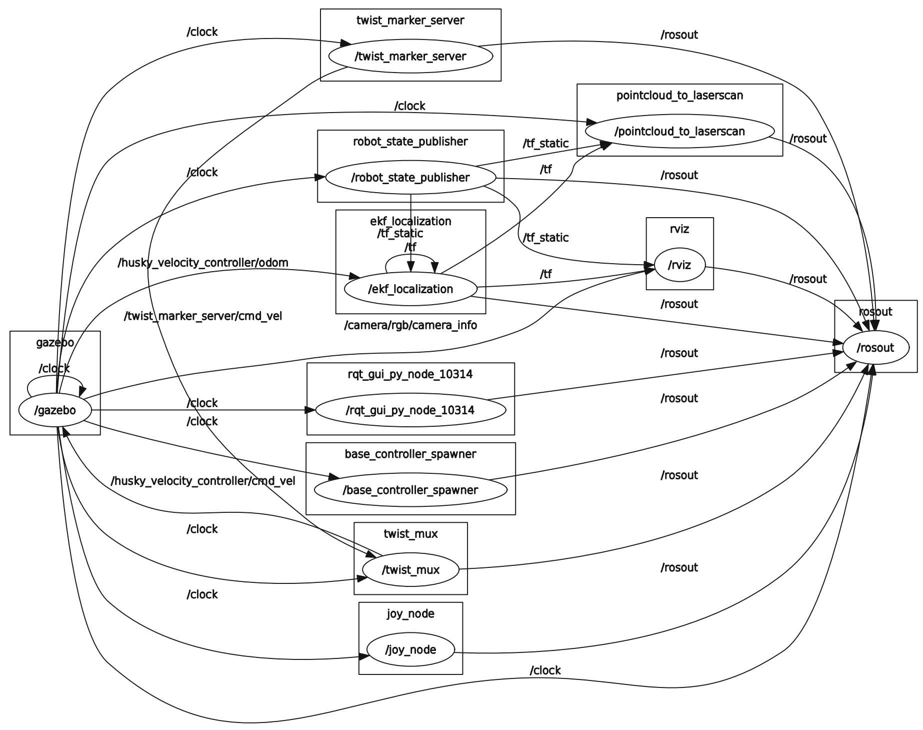
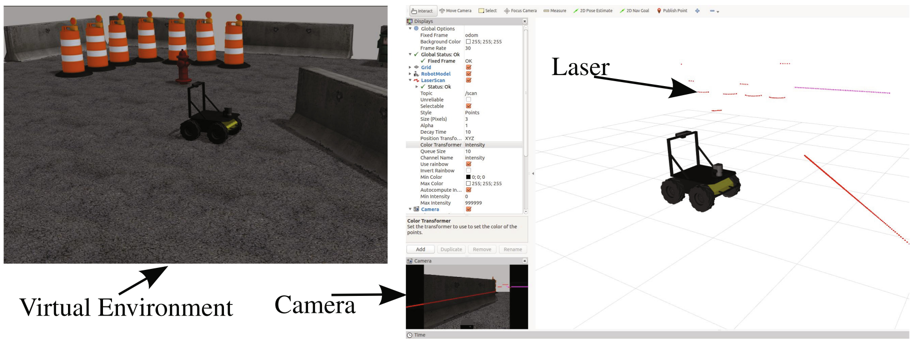


Fig. 2. An example of a ROS program. It is a graph where a node is a process providing or obtaining data from other processes. Related processes are linked by edges and denote interaction based on message passing.

it makes it possible to define a common communication channel between software components that control the behavior of the robot and the hardware components actually enforcing such a behavior. In this sense, ROS reduces the effort of tailoring software components to each particular kind of robot. For example, in ROS, any robot can be commanded to move by using the Twist message by indicating the linear and angular velocities. ROS assumes that each robot will execute this command accordingly to its own "anatomy". In the case of wheeled robots, only linear velocity has an important meaning since the angular velocity in the axis of movement is perpendicular to the floor. In the case of aerial vehicles, they can be commanded to change position by using three linear and three angular velocities, without any kinematic restriction.

5 Layered Architecture for the Integration

This section presents the main contribution of this paper. The key idea is to use PLEXIL as reactive layer and thus, by taking advantage of all automatic formal analysis techniques and tools available to it, enable the development of autonomous control modules for ROS with high degrees of reliability.



Global Options	
Fixed Frame	odom
Background Color	☐ 255; 255; 255
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
LaserScan	✓
✓ Status: Ok	
Topic	/scan
Unreliable	☐
Selectable	✓
Style	Points
Size (Pixels)	3
Alpha	1
Decay Time	10
Position Transfo...	XYZ
Color Transformer	Intensity
Queue Size	10
Channel Name	intensity
Use rainbow	✓
Invert Rainbow	☐
Min Color	■ 0; 0; 0
Max Color	☐ 255; 255; 255
Autocompute In...	✓
Min Intensity	0
Max Intensity	999999
Camera	✓

Color Transformer
Set the transformer to use to set the color of the points.

Fig. 3. Example of interaction between the Gazebo simulation environment and a Husky robot deployed in an environment with obstacles.

The proposed architecture is depicted in Fig. 4. It is based on the hybrid deliberative/reactive paradigm (see Sect. 2) and has the following elements:

The top layer (or deliberative layer): a software component that uses a global "environment model" (e.g., it contains terrain features and obstacles) and on a set of goals or problem description. It ultimately generates a mission plan composed of subtasks required to accomplish the goals in the given world model.

The deliberative/reactive binding layer: a software component responsible for providing compatibility between the format of the mission plan generated by the planner in the deliberative layer and the language used to describe the hierarchy of action nodes in the reactive layer.

The reactive layer: a hierarchical composition of nodes in which nodes at the top represent high-level behavior in the mission plan and the ones at the bottom represent basic actions in the plan (e.g., primitive robot commands and variable assignment).

The driver layer: a software component using ROS and the ROS drivers required for the specific robot components. This layer represents the hardware abstraction for the robot at the level of sensor actions and their access.

In particular, the control architecture resides in the reactive layer. On the one hand, this layer handles information from the environment such as changes on Yaw or position coordinates, which are updated based on data received via the driver layer from ROS drivers. On the other hand, the hierarchical composition of nodes in this layer has, at the top level, goal-oriented nodes such as driving straight for a distance or following a sequence of points. Such nodes will decompose in lower-level action nodes that, for instance, can perform tracking strategies such as follow-the-carrot or pure-pursuit [12] by calculating the required speed and steering settings.

As a proof of concept, a general purpose PLEXIL-ROS adapter has been developed by the authors, following the ideas proposed as future work in [3]. Conceptually, this adapter transforms ROS events triggered by the environment into

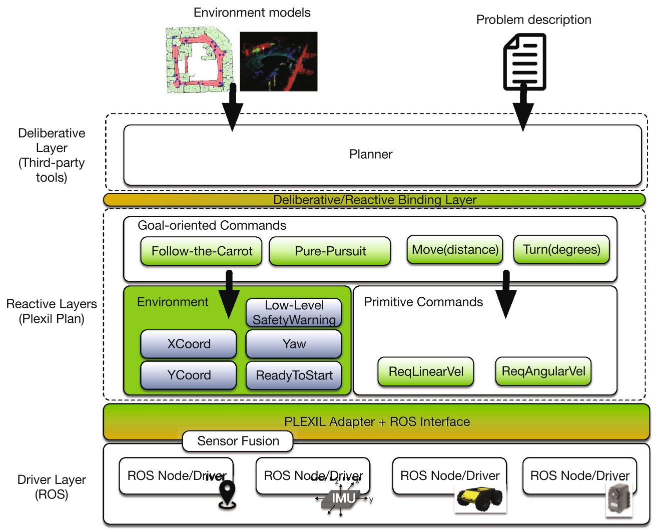


Fig. 4. Proposed ROS and PLEXIL layered architecture.

variables that PLEXIL plans can handle, and PLEXIL commands into requests to ROS driver nodes. Figure 5 depicts how the PLEXIL-ROS adapter integrates PLEXIL plans and ROS nodes, transforming the data that flows from the layers when necessary:

1. The PLEXIL Universal Executive executes the plan defined for the reactive layer of the architecture using a PLEXIL adapter.
2. The adapter defines the low-level operations and environment variables for such a plan.
3. The adapter uses an interface that is subscribed to relevant events in ROS.
4. The adapter transforms the generated events by updating environment variables and handles command invocation by publishing ROS events.
5. The message-oriented middle-ware of ROS (ROSCORE) enables the indirect communication with a real or simulated robot.

As a technical detail, it is important to note that in order to ensure that the PLEXIL plan in the reactive layer is able to check that the environment information has been initialized – preventing the initialization of the plan with inconsistent data –, a special variable called 'Ready' is assigned true when ROS reports the status of the robot for the first time. The current implementation of the adapter, for testing purposes, is based on the Husky platform [20].

6 Proof of Concept

This section presents a proof of concept of the implementation of the PLEXIL-ROS adapter proposed in Sect. 5. The Husky robot platform – via a Gazebo

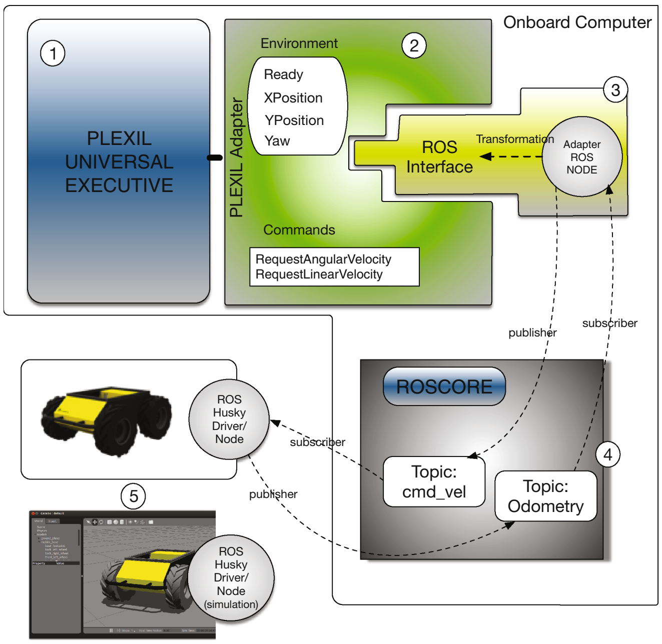


Fig. 5. Proposed PLEXIL-ROS integration architecture.

simulation – and a basic path tracking scenario were chosen for the experiment. The goal of the experiment is to demonstrate how a PLEXIL plan in the reactive layer can command a ROS-enabled robot whose drivers are in the hardware-abstraction layer. Note that for the purpose of this paper, the plan has been developed manually and without the help of a deliberative layer software.

The experiment is the following: make a Husky follow a fixed-size square-shaped path specified by a PLEXIL plan. Figure 6 depicts the logical description of the plan as a tree of tasks and Listing 1.1 presents the code of the plan. Intuitively, the PLEXIL plan consists of:

– Two high-level nodes: move forward for a given distance and turn a given amount of degrees.
– The plan performs four consecutive iterations of the sequence: move N meters, stop, and turn 90 degrees.
– The `Move` node is defined as a sequence of two nodes: `OdometryUpdate` and `MoveUntilDistanceReached`. The first one performs a lookup of the current position and the second one performs a `LinearVelocityRequest` until the repeat condition is not met (i.e., until the distance from the starting point to the current position is less than the expected distance).
– Angular velocity is continuously requested by `Stop`.
– Node `Turn` re-calculates the angular displacement every time a new Yaw is reported. Once the angular displacement is approximately close to the

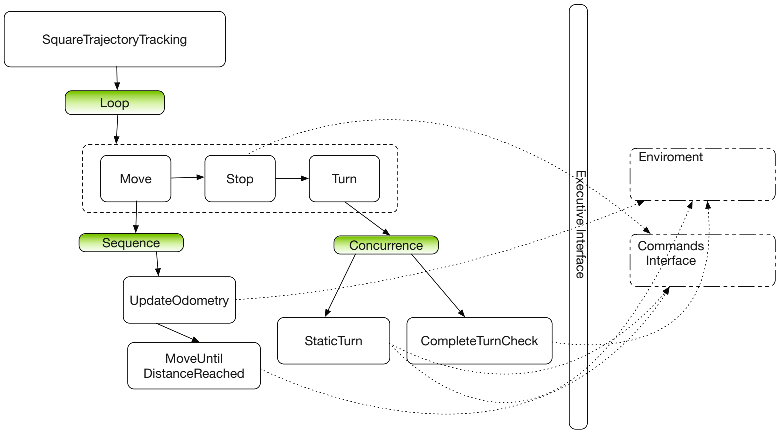


Fig. 6. PLEXIL plan tree for the proof of concept.

expected one, `CompleteTurnCheck` makes `StaticTurn` reach the final (i.e., FINISHED) state in the plan.

Listing 1.1. PLEXIL plan for the proof of concept.

```
Command RequestLinearVelocity(Real);
Command RequestAngularVelocity(Real);
Command pprint(...);
SquareTrajectoryTracking:{
  Boolean completeTurn=false, goal=false;
  Real PI=3.1416, initialYaw=0, WIDTH=5;
  Start Lookup(Ready);
  initialYaw=Lookup(Yaw);
  //four times: move forward, turn 90deg
  for (Integer i = 1; i <= 4; i + 1) {
    //Move until distance from starting position (aprox)== WIDTH
    Move:{
      Real currXPos,currYPos;
      Sequence{
        UpdateOdom:{
          currXPos=Lookup(XPosition);
          currYPos=Lookup(YPosition);
        }
        MoveUntilDistanceReached:{
          Start
              sqrt((currXPos-Lookup(XPosition))*(currXPos-Lookup(XPosition))+
               (currYPos-Lookup(YPosition))*(currYPos-Lookup(YPosition))) <
                  WIDTH;
          Repeat
              sqrt((currXPos-Lookup(XPosition))*(currXPos-Lookup(XPosition))+
               (currYPos-Lookup(YPosition))*(currYPos-Lookup(YPosition))) <
                  WIDTH;
           RequestLinearVelocity(1);
```

```
                }
            }
        }
        Stop:{
            RequestAngularVelocity(0);
        }
        //turn until a PI/2 rotation is achieved
        Turn:{
            initialYaw=Lookup(Yaw);
            completeTurn=false;
            Concurrence{
                StaticTurn:{
                    Repeat completeTurn==false;
                    RequestAngularVelocity(0.1);
                }
                CompleteTurnCheck:{
                    Start   (abs(initialYaw-Lookup(Yaw)) <= PI &&
                           abs(initialYaw-Lookup(Yaw))>=PI/2) ||
                            (abs(initialYaw-Lookup(Yaw)) > PI &&
                                  (2*PI-abs(initialYaw-Lookup(Yaw)))>=PI/2);
                    completeTurn=true;
                }
            }
        }
    }
```

Finally, Fig. 7 shows a superposition of screenshots of the simulation executed in Gazebo.

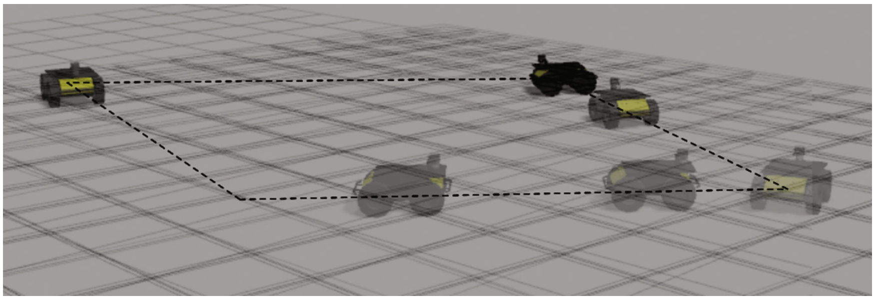


Fig. 7. Gazebo simulation.

7 Concluding Remarks

Designing and developing control software for an autonomous robot is a challenging and complex task, specially considering that – with cyber-physical systems – the costs of failure or unexpected behaviors can be dramatic. Unfortunately, formal verification methods, a powerful set of techniques and tools aimed at ensuring software correctness and reliability are not easily accessible to most robot developers. Although ROS has simplified the development of robotic solutions with its low level control layer commanding a robot to perform desired movements, moving the actuators (e.g., servo-motors), and process data provided by sensors, the problem of identifying control flaws is far from solved. Common flaws include deadlocks, violation of conformance to temporal, spatial, or timed constraints, which are key to mission-critical applications.

This paper has presented a software architecture and implementation for integrating PLEXIL and ROS. The main idea is to use PLEXIL, a synchronous reactive language created by NASA for mission critical robotic systems, in the reactive layer in a hybrid architecture to command ROS-enabled robots. PLEXIL has been proved to be mathematically reliable and robust by the scientific community. Given the fact that during the past 10 years ROS has become a de-facto standard in robotics, both in research and in industry, the PLEXIL-ROS integration can have positive impact in the development of mission-critical plans for autonomy in rovers. For example, it offers automatic verification techniques and tools already available for PLEXIL to ROS programmers. The proposed architecture has been validated and illustrated with an example consisting of an autonomous plan written in PLEXIL for a wheeled ROS-enabled robot.

Future work will integrate PLEXIL-compatible modules in the deliberative layer, which will be tested with specific tasks. They can include strategic areas in Colombia where reliable automation is needed. These include, for instance, precision agriculture. On the other hand, it will be ideal to deploy an official PLEXIL-ROS package to make the integration of components and the validation environments above-mentioned fully available to the ROS programming community. The feedback from the ROS community will be a valuable input for future stages of this project. Finally, specific-purpose verification techniques and tools need to be developed for the PLEXIL-ROS integration depending on the area of use.

References

1. Andres, B., Rajaratnam, D., Sabuncu, O., Schaub, T.: Integrating ASP into ROS for reasoning in robots. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS, vol. 9345, pp. 69–82. Springer, Cham (2015). doi:10.1007/978-3-319-23264-5_7
2. Broenink, J., Brodskiy, Y., Dresscher, D., Stramigioli, S.: Robustness inembedded software for autonomous robots. Mikroniek **54**, 38–45 (2014)
3. Cadavid, H.F., Chaparro, J.A.: Hardware and software architecture for plexil-based, simulation supported, robot automation. In: IEEE Colombian Conference on Robotics and Automation (CCRA), pp. 1–6. IEEE (2016)
4. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic. LNCS, vol. 4350. Springer, Heidelberg (2007)

5. Dowek, G., Muñoz, C., Rocha, C.: Rewriting logic semantics of a plan execution language. Electron. Proc. Theoret. Comput. Sci. **18**, 77–91 (2010)
6. Estlin, T., Jonsson, A., Pasareanu, C., Simmons, R., Tso, K., Verma, V.: Plan Execution Interchange Language (PLEXIL). Technical report TM-2006-213483, NASA, April 2006
7. O. S. R. Foundation. GAZEBO: A 3D dynamic simulator. http://gazebosim.org. Accessed 19 May 2017
8. O. S. R. Foundation. ROS: Robot operating system. http://wiki.ros.org. Accessed 19 May 2017
9. O. S. R. Foundation. RViz: 3D visualization tool for ROS. http://wiki.ros.org/rviz. Accessed 19 May 2017
10. Janssen, R., van Meijl, E., Di Marco, D., van de Molengraft, R., Steinbuch, M.: Integrating planning and execution for ros enabled service robots using hierarchical action representations. In: 2013 16th International Conference on Advanced Robotics (ICAR), pp. 1–7. IEEE (2013)
11. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, pp. 2149–2154, September 2004
12. Lundgren, M.: Path tracking for a miniature robot. Department of Computer Science, University of Umea, Masters (2003)
13. Medeiros, A.A.: A survey of control architectures for autonomous mobile robots. J. Braz. Comput. Soc. **4**(3) (1998)
14. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theoret. Comput. Sci. **96**(1), 73–155 (1992)
15. Muñoz, C.A., Dutle, A., Narkawicz, A., Upchurch, J.: Unmanned aircraft systems in the national airspace system: a formal methods perspective. SIGLOG News **3**(3), 67–76 (2016)
16. Muñoz, P., R-Moreno, M.D., Castaño, B.: Integrating a PDDL-based planner and a PLEXIL-executor into the ptinto robot. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J.M., Ali, M. (eds.) IEA/AIE 2010. LNCS, vol. 6096, pp. 72–81. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13022-9_8
17. Nakhaeinia, D., Tang, S.H., Noor, S.M., Motlagh, O.: A review of control architectures for autonomous navigation of mobile robots. Int. J. Phys. Sci. **6**(2), 169–174 (2011)
18. Potop-Butucaru, D., de Simone, R., Talpin, J.-P.: The synchronous hypothesis and synchronous languages. In: The Embedded Systems Handbook, pp. 1–21 (2005)
19. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, p. 5 (2009)
20. Robotics, C.: Husky-unmanned ground vehicle. Technical Specifications, Clearpath Robotics, Kitcener, Ontario, Canada (2013)
21. Rocha, C.: Symbolic Reachability Analysis for Rewrite Theories. Ph.D. thesis, University of Illinois, December 2012
22. Rocha, C., Cadavid, H., Muñoz, C., Siminiceanu, R.: A formal interactive verification environment for the plan execution interchange language. In: Derrick, J., Gnesi, S., Latella, D., Treharne, H. (eds.) IFM 2012. LNCS, vol. 7321, pp. 343–357. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30729-4_24
23. Rocha, C., Meseguer, J., Muñoz, C.: Rewriting modulo SMT and open system analysis. J. Logic. Algebr. Methods Program. **86**(1), 269–297 (2017)

24. Rocha, C., Muñoz, C., Cadavid, H.: A graphical environment for the semantic validation of a plan execution language. In: Third IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT 2009), pp. 201–207. IEEE, July 2009
25. Rozier, K.Y.: Specification: the biggest bottleneck in formal methods and autonomy. In: Blazy, S., Chechik, M. (eds.) VSTTE 2016. LNCS, vol. 9971, pp. 8–26. Springer, Cham (2016). doi:10.1007/978-3-319-48869-1_2
26. Verma, V., Jonsson, A., Pasareanu, C., Iatauro, M.: Universal-executive and PLEXIL: engine and language for robust spacecraft control and operations. In: American Institute of Aeronautics and Astronautics SPACE Forum (Space 2006). American Institute of Aeronautics and Astronautics, September 2006
27. Zheltoukhov, A.A., Stankevich, L.A.: A survey of control architectures for autonomous mobile robots. In: 2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), pp. 1094–1099. IEEE (2017)