PROMISE: High-Level Mission Specification for Multiple Robots

Sergio García
Chalmers | University of Gothenburg
Gothenburg, Sweden
sergio.garcia@gu.se

Patrizio Pelliccione
Chalmers | University of Gothenburg
Gothenburg, Sweden
University of L'Aquila
L'Aquila, Italy
patrizio.pelliccione@gu.se

Claudio Menghi
University of Luxembourg
Luxembourg, Luxembourg
claudio.menghi@uni.lu

Thorsten Berger
Chalmers | University of Gothenburg
Gothenburg, Sweden
thorsten.berger@gu.se

Tomas Bures
Charles University
Prague, Czech Republic
bures@d3s.mff.cuni.cze

ABSTRACT

Service robots, a type of robots that perform useful tasks for humans, are foreseen to be broadly used in the near future in both social and industrial scenarios. Those robots will be required to operate in dynamic environments, collaborating among them or with users. Specifying the list of requested tasks to be achieved by a robotic team is far from being trivial. Therefore, mission specification languages and tools need to be expressive enough to allow the specification of complex missions (e.g., detailing recovery actions), while being reachable by domain experts who might not be knowledgeable of programming languages. To support domain experts, we developed PROMISE, a Domain-Specific Language that allows mission specification for multiple robots in a user-friendly, yet rigorous manner. PROMISE is built as an Eclipse plugin that provides a textual and a graphical interface for mission specification. Our tool is in turn integrated into a software framework, which provides functionalities as: (1) automatic generation from specification, (2) sending of missions to the robotic team; and (3) interpretation and management of missions during execution time. PROMISE and its framework implementation have been validated through simulation and real-world experiments with four different robotic models.

Video: https://youtu.be/RMtqwY2GOlQ

1 INTRODUCTION

It is envisioned the increase of investment and inclusion of service robots in several market sectors (e.g., logistics, medical, or farming) in the next few years.[1] As the number of deployed service robots deployed increases, humans will likely have to interact with robots in their everyday life. Imagine for instance a near-future social or industrial environment (e.g., a hotel, a hospital, or a warehouse) where teams of service robots are deployed to achieve tasks collaboratively. Those tasks will be configured and specified by domain experts and end-users who may not be knowledgeable on programming languages, according to the H2020 Robotics Multi-Annual Roadmap (MAR).[2] According to this roadmap, this will require: (1) that robots "need to become intuitively integrated with human operators", (2) "highly abstracted mission definition [...] algorithms for interaction and operation with untrained users." In fact, these requirements are confirmed by two recent studies [9, 10], whose authors claim that precisely specifying missions and transforming them for automatic processing are one of the main challenges in robotics software engineering. Therefore, the current state of robot mission specification methods and tools must become friendlier in their usage to the user. We aim to promote the simplicity of our tool while keeping its expressiveness. At the same time, we strive to enable a rigorous specification that describes precisely and unambiguously the mission the robot must perform. As explained in a recent study [8], existing solutions fail at providing a trade-off of those qualities.

In this paper, we present a framework that supports users in mission specification for multi-robot applications. The framework integrates PROMISE (simPle RObot MIssion SpEcification), a language integrated as a DSL. The DSL permits the specification of complex missions (i.e., complex behaviors, including recovery actions) while being accessible for users not knowledgeable of programming languages. Moreover, PROMISE builds upon a catalog of mission specification LTL-based patterns [10], ensuring the correctness of the semantics and therefore the rigor of the tool. The framework also integrates software components for the automatic mission generation, sending, and interpretation and management. We detail the framework implementation, workflow, and evaluation of PROMISE through experimentation.

Running example. We use an example inspired by a milestone experiment from a European project[3] with which our DSL's development is involved to illustrate different aspects throughout the paper. The example represents two robots collaboratively working in an industrial warehouse. A mobile platform $r1$ patrols a set of locations (assembly stations). Users may use the robot to deliver items to other stations

[1] www.ifr.org

[2] https://eu-robotics.net

[3] http://www.co4robots.eu/

and can stop the patrolling using a specific gesture. $r1$ continuously evaluates if station 2 ($l2$) has assembled a product (which is recognizable by the robot). If $r1$ detects such a product, it requests help to $r2$. The mobile manipulator $r2$ has been waiting in an idle state now moves to $l2$ and loads the object on top of $r1$. If the loading task fails, $r2$ communicates the failure through its speakers.

Users. We now categorize the users of our framework.

Domain expert. User characterized as a worker from one of the scenarios previously described referring to the MAR. This user might be knowledgeable of robotics but not of programming languages. Their main interaction with PROMISE is to specify missions (as the one shown in Fig. 1) by instantiating actions that were previously implemented by *developers*.

Developer. User with programming knowledge in charge of developing new robotic skills and actions to the tool. Referring to the running example, a possible use case for the developer is to develop and implement robotic actions, as the ones that permit the robots recognize specific gestures from a human.

Maintainer. This user is in charge of performing maintenance services over the whole framework and its implementation. We envision a use case for the maintainer: to assure the correct functioning of the framework and its implementation with other solutions that may provide robotic functionalities as planning or motion control.

2 RELATED WORK

Petri Nets [14] and Statecharts [1] are solutions for mission specification. As opposed to our goal, these solutions rely on an imperative, step-by-step description of a robot's mission. DSLs are a great approach to solve domain-specific problems [12], such as mission specification in robotics. Various DSLs have been proposed for modeling robotic systems and their missions [11]. However, most of the effort has been made for the domain of industrial robots, a field where users are often specialized and where robots work in controlled environments not populated by humans. On the contrary, human-populated environments are typically dynamic and unconstrained, enforcing robotic systems to be able to react to events to provide some level of robustness and adaptation.

Many robotic companies have developed mission specification mechanisms, which are made available for their robots as IDEs[4] and frameworks [11]. Yet, those mechanisms are often platform- and robot-dependent. Among the existing mission specification proposals that are not developed or are constrained to a specific robot, many require programming skills from the user [6, 13]. Contrarily, our goal is to promote the simplicity of our language to make it accessible by end-users and domain experts. Some works also aimed at providing user-friendly solutions, like the one proposed by Bozhinoski et al. [2]—extended by Di Ruscio et al. [4]. Their tool allows non-technical users to define high-level specifications of missions for teams of multi-copters. However, the user is required to precisely detail the robots' behavior while we aim to provide a declarative specification where the user needs only to specify a high-level goal instead of the intermediate steps required to achieve it. Finally, Doherty et al. [5] propose a high-level mission-specification language for collaborative multi-robot applications that supports abstract yet complex and declarative mission specification by composing tasks using operators. However, their catalog of composition operators is limited and their application targets multi-copter vehicles, while we focus on ground service robots.

[4]What is Choregraphe?

3 PROMISE

PROMISE allows the definition of *global missions* to be achieved by a team of robots and its manual decomposition into *local missions*, i.e., robot-specific missions [7]. Our language builds upon a recently proposed catalog of mission specification LTL-based patterns [10]. These patterns are used as atomic tasks that may be composed utilizing a set of operators we proposed in our previous work [8] and which are inspired on behavior tree operators [3]. We conceived two types of operators: (1) composition operators allow the definition of complex missions by combining other operators and managing events that may occur in the environment; (2) delegate operators make a robot perform a specific task instantiated with parameters as locations or actions. In turn, each task translates to an LTL formula (e.g., from the running example, the task *Sequenced patrolling* $l1$, $l2$, $l3$ performed by $r1$ translates to $\mathcal{G}\left(\mathcal{F}\left(l1 \wedge \mathcal{F}\left(l2 \wedge \mathcal{F}\ l3\right)\right)\right)$.

This specific task makes the robot visit infinitely often a set of locations one after the other. These formulas are used by an underlying planner, which uses a model of the environment and of the robots to compute a plan—a set of low-level actions that, if performed, allows the achievement of the given mission.

PROMISE provides a graphical and a textual syntax, each of which is supported by a dedicated editor. The graphical syntax maps mission specification concerns to graphical elements that can be understood by users not knowledgeable on programming languages. Using this syntax, operators and tasks can be just *dragged&dropped* from a palette and interconnected accordingly or specified textually.

Our DSL defines missions that can react to external events, permitted by the semantics of certain operators. Figure 1 shows the graphical definition of a mission, the circled numbers were added to improve readability.

Node ① decomposes the global mission into two local missions, ② for $r1$ and ⑤ for $r2$. Note that the ordering of every operator is textually represented in its label (i.e., *default* and o_i). Node ② represents an operator *event handler*, which takes as input a set of operators and events. Operators and events are associated with other operators in a parent-child structure through edges—represented in Fig. 1 with gray arrows. When executed, the operator starts by performing its default mission, that is, the operator labeled as *default*. The execution of the event handler will not end until the execution of the default mission is finished (either succeeding or failing). In the example, the default mission of $r1$ is to sequentially patrol $l1$, $l2$, and $l3$ (③). PROMISE also allows the definition

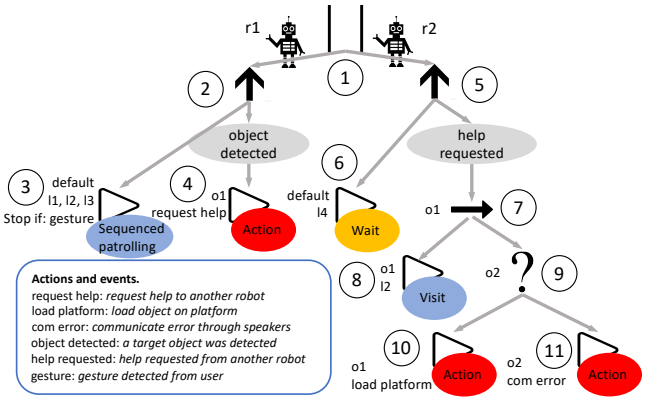

Figure 1: Mission specification using the graphical syntax (running example).

of events that if detected, will stop the execution of a task. If a delegate operator is configured with such an event, it will display in its label the string "Stop if: *event*". In this case, if $r1$ detects a specific gesture from a user it will stop its patrolling (③), which otherwise would go on forever. The event handler also is associated with behaviors that are triggered by events represented as gray ellipses. Thus, if *object detected* occurs and it is sensed by $r1$ during its default mission (③), ④ is triggered and $r1$ will perform the action *request help*. This specific action sends a message requesting help to a specific robot ($r2$). The default mission of an event handler is resumed as soon as the execution of one of its event-triggered behaviors is finished. An excerpt from the mission specification using the textual syntax is shown in Fig. 2, and it corresponds to the instantiation of the event handler ② and its children.

Meanwhile, $r2$ waits by default in $l4$ (⑤ and ⑥), and if during this idle state the robot receives a request of help, it will perform a set of tasks in a sequential manner (⑦), starting from moving to $l2$ (⑧). Then, the operator *fallback* (⑨) starts by executing its first child (⑩), i.e., load the mobile platform $r1$ with a target object. If this task fails (e.g., the object falls from the robot's gripper), the operator fallback dictates that the following child (⑪) is executed, and so on. With ⑪, the robot must communicate that a failure has occurred and then resume its waiting state in $l4$.

4 SOFTWARE FRAMEWORK

The software framework (Fig. 3) into which PROMISE is integrated consists of four elements distributed into two units. The unit *Central station* provides an interface to the user (typically deployed into a computer) and integrates elements ⓐ, ⓑ, and ⓒ.

```
eventHandler(
    default(
        delegate (SequencedPatrolling locations l1, l2, l3 stoppingEvents finish))
    except found_object (
        delegate (SimpleAction actions request_help)))
```

Figure 2: Textual syntax excerpt.

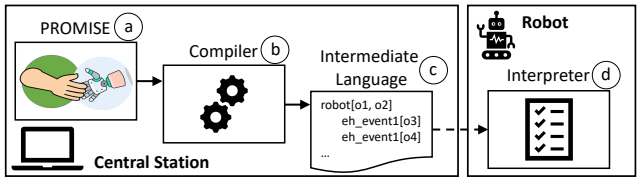

Figure 3: PROMISE and its software framework.

The unit *Robot* represents what is deployed and executed at run-time within each robot. Our framework builds upon an existing robotic platform and architecture, SERA [7]. This platform's current implementation relies on ROS[5] and provides a set of functionalities, including motion control, collision avoidance, image recognition, self-localization, and planning. Nevertheless, PROMISE is developed as a standalone tool and could be integrated with various robotic tools and platforms.

In Fig. 3, the component ⓐ encapsulates the language and DSL of PROMISE, realized as a plugin for Eclipse, using Xtext[6] for the textual interface and Sirius[7] for the graphical one. The compiler (component ⓑ) contains a script that automatically generates the local missions to be sent to each robot. This component also generates a description of the specified mission using natural English, which might help users while specifying missions to evaluate whether the description corresponds to what they wanted to express.

The intermediate language (component ⓒ) describes the set of tasks to be performed by each robot. The used software platform currently implements an LTL-based planner, so PROMISE's intermediate language is composed of a set of LTL formulae with the addition of the used operators' semantics. The intermediate language permits decoupling the mission specification from the robotic platform and the development of interpreter tools [8]. Finally, an interpreter (ⓓ) is deployed within each robot. This interpreter receives the robot-specific local mission and communicates the tasks to the local planner appropriately. In this way, PROMISE becomes robot-agnostic since only the robot-specific interfaces of the interpreter with the lower-level components of the platform must be adapted when using a new robot.

5 PROMISE IN ACTION

In this section, we present the workflow of PROMISE. The first steps correspond with the specification of the mission. The seamless integration of Sirius with Xtext allows the mission specification using the previously explained syntaxes in parallel. The typical workflow begins by using a wizard that supports users creating the basic structure of the mission, i.e., names of the used robots, locations, and conditions of the mission. Once finished with the wizard, a basic structure of the mission will spawn in the textual syntax. The user may continue using the textual syntax (writing the mission specification in a Java-like fashion), or using the graphical

[5] https://www.ros.org
[6] https://www.eclipse.org/Xtext/
[7] https://www.eclipse.org/sirius/

syntax dragging and dropping elements from the provided palette, or using both seamlessly. We put special emphasis on the development of the graphical syntax, since our goal is to support users and domain experts without deep programming skills. This syntax displays helpful messages in the labels of the operators and interconnecting edges. For instance, labels display the order of children operators and their configurations (i.e., actions or locations set as properties to them). The labels may also warn the user when operators have not been configured or remain unlinked. All these mission specification mechanisms are encapsulated in component ⓐ of Fig. 3.

Whenever the user saves a mission specification, the compiler (component ⓑ of Fig. 3) automatically generates two files for each used robot. One file encodes the set of LTL formulae that describe the tasks of each local mission as an intermediate language (ⓒ). The second file contains a description of the local mission in natural English. This description has proved to be useful for users to evaluate whether their concerns have been properly specified [8].

Once the user is satisfied with the specification they shall proceed to send it to the robots. We provide a Java script compiled into a jar file, which encodes the local missions and forwards them using services to the target robots (the IP of each robot must be specified). An interpreter (ⓓ) is deployed into each robot of the team. The interpreter receives the local mission and parses it. Finally, the interpreter uses the ROS environment to feed the underlying planner with tasks as requested. For instance, the interpreter of $r1$ firstly feeds its planner with the task *patrol* $l1$, $l2$, and $l3$. If the robot detects a recognizable object, this task is stopped and a new task is sent to the planner (in this case, *perform action request help*). Once the task is finished, the interpreter requests the planner to resume the stopped default mission. The interpreter displays useful information for the user through a terminal, for instance, the mission it has been received, the task which is being executed by the robot, the state of such task, detected events and whether they trigger any task of the local mission.

6 EXPERIMENTATION

We evaluate our DSL and implementation framework through experimentation, and with this aim, we integrated them into an existing platform that provides different robotic functionalities (e.g., planning, motion control, self-localization). The experimentation serves us to evaluate the workflow described in Sec. 5 and the components shown in Fig. 3. We provide videos of our experiments with different robots and environments in a dedicated website.[8]

We replicated missions which are available on literature, specifically from the RoboCup@Home 2018 competition.[9] The restaurant-management scenario was performed first in simulation and then in real environments. The robots we used for the experimentation of this scenario were a Turtlebot2[10] in the offices of the University of Gothenburg and a TIAGo robot in the facilities of PAL Robotics[11]. The repository with the implementation of our DSL[12] contains also the specification of two other missions from that edition of RoboCup@Home: the *dishwasher challenge* and the *tour guide*.

We evaluated our language simulating two other missions. The first is used as a running example in our previous work [8], and the second is represented in Fig. 1. Both comprise two robots performing collaborative and reactive tasks.

Finally, it is worth to mention that we validated the simplicity and expressiveness of PROMISE and its implementation through two user studies (more details are available in our previous publication [8]).

7 CONCLUSION

In this paper, we present a software framework that supports end-users and domain experts in multi-robot mission specification and execution. The framework integrates different software components, including a language realized as a DSL, a compiler, and an interpreter. We strive to keep the balance between simplicity and expressiveness in our language. We report the usage of the framework and its components by listing the possible stakeholders that may use it and illustrating use cases. We also introduce a running example, which is specified using PROMISE's graphical syntax. Finally, we show real examples of mission specification and execution in real and simulated environments using PROMISE and its framework.

REFERENCES

[1] J. Bohren and S. Cousins. 2010. The SMACH high-level executive [ROS news]. *IEEE Robotics & Automation Magazine* (2010).
[2] D. Bozhinoski, D. Di Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli. 2015. FLYAQ: Enabling Non-expert Users to Specify and Generate Missions of Autonomous Multicopters. In *ASE'15*.
[3] Michele Colledanchise and Petter Ögren. 2018. *Behavior Trees in Robotics and Al: An Introduction.* CRC Press.
[4] D. Di Ruscio, I. Malavolta, P. Pelliccione, and M. Tivoli. 2016. Automatic generation of detailed flight plans from high-level mission descriptions. In *MODELS'16*.
[5] P. Doherty, F. Heintz, and J. Kvarnström. 2013. High-level mission specification and planning for collaborative unmanned aircraft systems using delegation. *Unmanned Systems* (2013).
[6] P. Doherty, F. Heintz, and D. Landén. 2010. A distributed task specification language for mixed-initiative delegation. In *PRIMA'10*.
[7] S. García, C. Menghi, P. Pelliccione, T. Berger, and R. Wohlrab. 2018. An Architecture for Decentralized, Collaborative, and Autonomous Robots. In *ICSA'18*.
[8] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures. 2019. High-level mission specification for multiple robots. In *SLE'19*.
[9] S. Maoz and J. Ringert. 2018. On the software engineering challenges of applying reactive synthesis to robotics. In *RoSE'18*.
[10] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger. 2019. Specification Patterns for Robotic Missions. *TSE* (2019).
[11] A. Nordmann, N. Hochgeschwender, and S. Wrede. 2014. A survey on domain-specific languages in robotics. In *SIMPAR'14*.
[12] Douglas C. Schmidt. 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39, 2 (2006).
[13] D. Castro Silva, P. Henriques Abreu, L. Paulo Reis, and E. Oliveira. 2014. Development of a Flexible Language for Mission Description for Multi-robot Missions. *Information Sciences* (2014).
[14] V. Ziparo, L. Iocchi, D. Nardi, P. F. Palamara, and H. Costelha. 2008. Petri net plans: a formal model for representation and execution of multi-robot plans. In *AAMAS'08*.

[8] https://sites.google.com/view/promise-dsl/home
[9] http://www.robocupathome.org/rules
[10] https://www.turtlebot.com/turtlebot2/
[11] https://www.pal-robotics.com/
[12] https://github.com/SergioGarG/PROMISE_implementation