ROSSMARie: A Domain-Specific Language To Express Dynamic Safety Rules and Recovery Strategies for Autonomous Robots

Momina Rizwan[1], Christoph Reichenbach[1] and Volker Krueger[1]

Abstract— **Ensuring functional safety is a critical challenge for autonomous robots, as they must operate reliably and predictably despite uncertainty. However, existing safety measures can over-constrain the system, limiting the robot's availability to perform its assigned task. To address this problem, we propose a more flexible strategy that equips robots with the ability to adapt to system failures and recover from those situations without human intervention. We extend a domain-specific language, Declarative Robot Safety (DeROS), whose runtime stops a robot whenever it violates a safety rule (e.g., proximity to a human). Our extended language, ROSSMARie, adds the capability to monitor whether a rule is no longer violated and to recover and resume robot operation. We validate ROSSMARie on the ROS-based industrial platform SkiROS2 and verify its effectiveness in achieving safety and availability. Our experiments demonstrate that our DSL extension ensures functional safety while enabling robots to complete their tasks.**

I. Introduction

Ensuring the safe behavior of robots in a dynamic and unpredictable environment where humans are present is a challenging task. Autonomous robots must be designed to operate effectively in uncertain environments and are able to adapt to system failures without external interference [1]. While deploying autonomous robots in real-life settings, the safety of the environment, human users, and the robot itself is of prior importance.

Runtime monitoring is one way of guaranteeing safety which has been explored in recent years [2]. A safety monitor is an independent component that can detect potential safety violations and intervene with recovery or corrective strategies. Adam et. al [3] trigger a stop action when the robot encounters an unexpected situation and waits for the operator to start the robot. While this strategy may be suitable for their experiments, it is a *conservative* strategy that for an autonomous robot For example, in the case of service robots operating in households, if the safety protocol requires the robot to stop whenever it encounters uneven terrain, it may not be able to complete its task, such as cleaning a room, as efficiently as it could if it were able to slow down until it has crossed the uneven area and then continue cleaning at its normal speed.

While stopping the robot is an important safety measure to prevent accidents, it can hinder the robot's ability to function effectively in certain contexts. Therefore, it is important to strike a balance between safety and functionality when designing safety strategies [4]. Addressing these challenges requires innovative approaches to robot design and programming, as well as comprehensive safety protocols to ensure that robots can operate safely in a variety of contexts. By continuing to improve the safety behaviours of autonomous robots, we can unlock the full potential of these machines to make our lives easier and more efficient.

In this extended abstract, we focus on *functional safety* and assume that the sensors are reliable. A system is considered functionally safe if it operates correctly in response to its inputs and if it can't, then it should fail in a predictable manner [3]. We extend an existing domain-specific language designed in the spirit of DeROS [3] which generates a safety monitor (for ROS-based software) that enforces rules expressed in that language. The contributions include:

1) We modify the semantics of the language, enabling robots to resume after a hazardous/unsafe situation has passed.
2) We introduce strategies to recover from those safety hazards
3) We integrate the framework proposed by Adam et. al. [3] with SkiROS2, a skill-based platform for ROS, and we demonstrate the effectiveness of our recovery strategies by applying them to new scenarios.

To distinguish between the previous work and our implementation, we refer to our modified DSL as ROSSMARie.

II. Background: DeROS

DeROS [3] is a Domain-Specific Language (DSL) to express dynamic safety rules for runtime monitoring based on informal safety specifications that provide information on components in Robot Operating System (ROS) (topics and nodes). The DeROS compiler then generates a runtime safety monitor to check the specified properties of the software system. This framework proposed by Adam et. al. [3] is aimed at isolating safety handling from the robot functionality and treating it as a cross-cutting concern.

III. ROSMARie

ROSSMARie is an extension of DeROS with modified semantics. The runtime semantics of ROSSMARie enable continuous feedback from the sensors and allows the robot to resume its current operation as soon as the sensor values are in a safe range. We have implemented ROSSMARie in the JastAdd [5] meta-compiler.

A. Integration with SkiROS2

SkiROS2 [6] is a skill-based robot control platform that can execute multiple skills st the same time with the help

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

[1]Department of Computer Science, Faculty of Engineering (LTH), Lund University, SE 221 00 Lund, Sweden. E-mail: <firstname>.<lastname>@cs.lth.se.

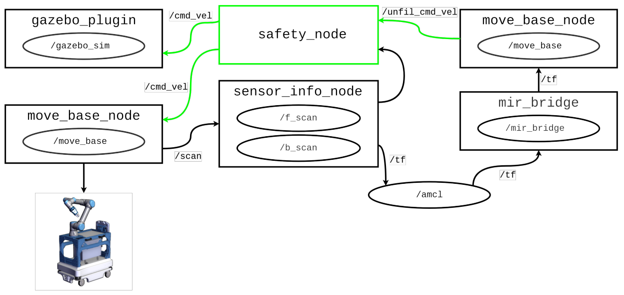

Fig. 1. Safety monitor acting as a filter to avoid conflicting information published on a topic.

of action server-based communication. Actions provide non-blocking background processing and are ideal for executing longer robot skills. While integrating ROSSMARie with SkiROS2, we encountered a problem where different modules send (publish) conflicting messages on the same ROS topic. This can result in jittering or unsafe behaviours from the robot. We realized that action servers need special handling. To address this, we added a *safety node* to choose which message to publish on the topic. Figure 1 shows an example of a safety node that filters out commands that set velocity. To introduce such a filter, we use the remap function in roslaunch. Remapping redirects a ROS node to publish on /unfil_cmd_vel instead of/cmd_vel.

To support ROS action servers in ROSSMARie code generation, we cancel the previous goal and send a new goal to the action server. DeROS, in comparison, can only support components that communicate using topic-based publish-subscribe communication.

IV. Experiments

To showcase the recovery strategies, we performed experiments with a robot named Heron, as shown in Figure 2. Heron integrates a MIR 100 with a UR5e. The MIR 100 is an indoor autonomous mobile robot with a maximum payload capacity of 100 kg. It has two laser scanners and ultrasonic sensors providing 360° visual feedback. The maximum speed of the robot is 1.5 m/s forward and 0.3 m/s backward. The UR5e is a 20.6 kg robot arm with a maximum payload capacity of 5 kg. It has a six-axis force/torque sensor to detect collisions. We ran our experiments using SkiROS2 skills.

Case study I: We used ROSSMARie to define safety rules for Heron navigating in uneven terrain with three safety hazards i.e. bumps, slight and steep ramps shown in Figure 2(a). If the robot encounters a bump with one wheel (detected through an IMU sensor), our rules reduce the speed until the robot has fully crossed the bump. On the other hand, a slight ramp triggers an increase in speed to enter the operating area. If the ramp is steep, the robot recovery strategy is to go backward and replan. In all three scenarios, the robot tried to recover from the situation that could have led to damage to the robot.

Case Study II: Figure 2(b) shows a simulation setup where Heron's task is to move an object from one table and place it on another. An active compliance controller can produce vibrations in the robot arm when contact occurs

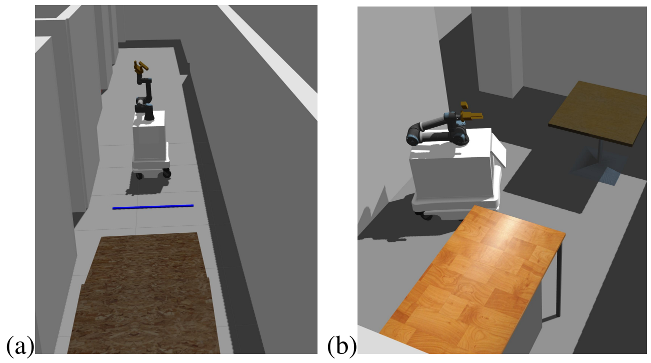

Fig. 2. (a) Bump and a ramp in the corridor. (b) Placing a block on the unexpectedly high table with an unknown height.

while placing the object. To avoid any serious damage, we switch to position control whenever the force torque sensor detects those vibrations.

Case Study III: Proactive behaviours are required to avoid deadly/costly repercussions e.g. harming a human in a crash is more costly. For human-shared workspaces, we defined a safety rule to decrease arm speed whenever a human is detected in the room.

V. Limitations and Future Work

While conducting our experiments, we observed that recovery strategies can vary depending on the task at hand. While safety rules typically aim to maintain a safe distance from objects, some tasks require interacting with (e.g., pushing) an object. In such interaction scenarios, the robot may need to approach the object more closely than safety rules would normally allow. We plan to address such scenarios by allowing ROSSMARie's safety rules to be task-aware. During experiments, we also encountered conflicting recovery strategies for different safety rules. To identify and resolve such conflicts, we plan to statically check rules for overlap.

References

[1] M. Müller, T. Müller, B. Ashtari Talkhestani, P. Marks, N. Jazdi, and M. Weyrich, "Industrial autonomous systems: a survey on definitions, characteristics and abilities," *at-Automatisierungstechnik*, vol. 69, no. 1, pp. 3–13, 2021.

[2] L. Masson, J. Guiochet, H. Waeselynck, K. Cabrera, S. Cassel, and M. Törngren, "Tuning permissiveness of active safety monitors for autonomous systems," in *NASA Formal Methods Symposium*, pp. 333–348, Springer, 2018.

[3] S. Adam, M. Larsen, K. Jensen, and U. P. Schultz, "Rule-based dynamic safety monitoring for mobile robots," *Journal of Software Engineering for Robotics*, vol. 7, no. 1, pp. 121–141, 2016.

[4] J. Arents, V. Abolins, J. Judvaitis, O. Vismanis, A. Oraby, and K. Ozols, "Human–robot collaboration trends and safety aspects: A systematic review," *Journal of Sensor and Actuator Networks*, vol. 10, no. 3, p. 48, 2021.

[5] G. Hedin and E. Magnusson, "Jastadd—an aspect-oriented compiler construction system," *Science of Computer Programming*, vol. 47, no. 1, pp. 37–58, 2003.

[6] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. Petrick, and V. Krüger, "Skiros—a skill-based robot control platform on top of ros," in *Robot Operating System (ROS)*, pp. 121–160, Springer, 2017.