The rclc Executor: Domain-specific deterministic scheduling mechanisms for ROS applications on microcontrollers: work-in-progress

Jan Staschulat
Corporate Research
Robert Bosch GmbH
Renningen, Germany
jan.staschulat@de.bosch.com

Ingo Lütkebohle
Corporate Research
Robert Bosch GmbH
Renningen, Germany
ingo.luetkebohle@de.bosch.com

Ralph Lange
Corporate Research
Robert Bosch GmbH
Renningen, Germany
ralph.lange@de.bosch.com

***Abstract*— Robots are networks of a variety of computing devices, such as powerful computing platforms but also tiny microcontrollers. The Robot Operating System (ROS) is the dominant framework for powerful computing devices. While ROS version 2 adds important features like quality of service and security, it cannot be directly applied to microcontrollers because of its large memory footprint. The micro-ROS project has ported the ROS 2 API to microcontrollers. However, the standard ROS 2 concepts are not enough for real-time performance: In the ROS 2 release "Foxy", the standard ROS 2 Executor, which is the central component responsible for handling timers and incoming message data, is neither real-time capable nor deterministic. Domain-specific requirements of mobile robots, like sense-plan-act control loops, cannot be addressed with the standard ROS 2 Executor. In this paper, we present an advanced Executor for the ROS 2 C API which provides deterministic scheduling and supports domain-specific requirements. A proof-of-concept is demonstrated on a 32-bit microcontroller.**

Keywords*— *real-time operating systems, scheduling, microcontrollers, robotics

I. Introduction

Robots, like many other cyber-physical-systems, are networks of computing devices. Microcontrollers are usually used to connect sensing and actuation and to realize low-level control. Here, hardware access and particularly deterministic real-time computation are important, e.g., for safe obstacle avoidance. More powerful computing devices ("hosts") execute higher-level functions where processing power is important, e.g., path planning and mapping.

Because of very different resources available, microcontrollers and host devices usually run very different base software, which leads to conceptual gaps, integration effort, and, arguably, functionality not always placed on the most appropriate device.

Some attempts at deeper integration exist: For example, rosserial [1] is a library for microcontrollers to interoperate with the Robot Operating System (ROS) on the host side. However, it is a separate implementation with very limited features. Approaches such as mROS [2] go further, but at far higher cost in terms of resources (e.g., mROS requires a 400 MHz MCU with 10MB of RAM). Both of these also only support ROS version 1.

For ROS version 2 (ROS 2), which adds important features such as quality of service and security, the micro-ROS project [3] has ported the standard stack to medium-sized microcontrollers (~100KB RAM). This means that all standard concepts, such as topics, services, parameters, lifecycle, actions, etc. are available with the same API as on the host, and that while resource constraints are still a factor, porting has become much less effort.

However, the standard concepts are not enough to achieve the performance we require from microcontrollers. In particular, the standard ROS 2 Executor, which is the central component responsible for handling timers and incoming message data, is neither real-time capable nor deterministic [4] in all current ROS 2 releases, up to and including the most recent "Foxy" release. Domain-specific requirements of mobile robots, like sense-plan-act control cannot be met and must be implemented in the application.

Therefore, we present in this work an advanced Executor for the ROS 2 C API (rclc) which provides deterministic execution and supports domain-specific requirements.

II. micro-ROS architecture

Figure 1 depicts the architecture of micro-ROS. The three main changes compared to standard ROS 2 are: First, the use of an RTOS (Zephyr, FreeRTOS and NuttX are currently supported) instead of a desktop operating system; second, the use a DDS-XRCE implementation instead of normal DDS and, third, the extension of the ROS Client Support Library rcl by the rclc package [5] to make it a full C API.

More resource demanding features, like node discovery, are implemented on the ROS 2 agent, which runs on a host with standard ROS 2

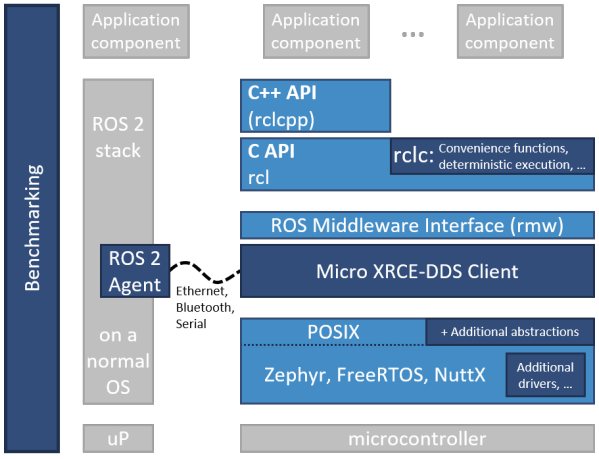

Figure 1: micro-ROS Architecture

Published in Proc. of Int'l Conf. on Embedded Software (EMSOFT), pp. 18-19. 09/2020. DOI: 10.1109/EMSOFT51651.2020.9244014

More resource demanding features, like node discovery, are implemented on the *ROS 2 agent*, which runs on a host with standard ROS 2.

III. RCLC EXECUTOR

A. Domain-specific requirements

Mobile robots often have domain-specific requirements, derived from the application. For example sense-plan-act control loops (Figure 2), synchronization of sensor data with different rates or prioritized processing (e.g., for safe obstacle avoidance). One requirement for a high quality localization is that all received data from multiple sensors should be processed before the localization algorithm runs itself. This leads to the design pattern of a phased-execution.

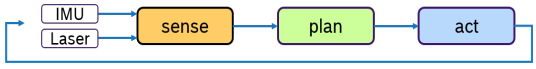

Figure 2: Sense-plan-act control loop

Deterministically synchronizing sensor data with different rates is hard, due to latency jitter and clock drift. An alternative is to wait for the input of one sensor (e.g. Laser) and then request all other sensors (e.g. IMU) to send their data, as shown in Figure 3. This requires to specify a pre-defined

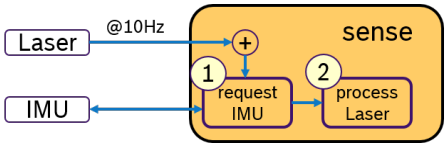

Figure 3: Sensor data synchronization with a trigger

processing order of callbacks in the Executor.

B. Deterministic scheduling

Based on the analysis of domain-specific requirements, the rclc Executor provides two main features [5]: First, the user can specify a sequential processing order of all callbacks. A callback is the corresponding function which is called on incoming messages, timer events, or hardware events. Second, a trigger condition defines when the processing starts. The following predefined trigger conditions are available: one, any and all. They trigger if one particular, at least one or all callbacks are ready for execution, respectively. Additionally, the user can define a custom trigger.

As memory is limited on a microcontroller, dynamic memory is only allocated at the startup of the rclc Executor. The performance overhead for this static scheduling with sequential processing is minimal compared to the overhead of the current ROS 2 Executor.

C. Demonstrator

The micro-ROS Kobuki demo [6] illustrates the use of micro-ROS and the rclc Executor on the Kobuki platform,

Figure 4: micro-ROS Kobuki demo

which is the mobile base of the well-known Turtlebot 2 research robot. Instead of a laptop running ROS 2, the Kobuki is equipped with a STM32F4 microcontroller only. This STM32F4 runs the micro-ROS stack and a port of the thin_kobuki driver [7] interacts with the robot's firmware using the rclc Executor. This setup is depicted in Figure 4.

The STM32F4 communicates the sensor data to a remote laptop running a standard ROS 2 stack. At the same time, using the other direction of communication, the Kobuki can be remote-controlled.

The trigger functionality has been demonstrated by synchronizing two input data streams with different rates. This example can be downloaded from the rclc_examples package in [5].

IV. CONCLUSION

The paper presented the rclc Executor, which supports domain-specific requirements and provides deterministic scheduling for ROS 2 applications on microcontrollers. A proof-of-concept demonstrator has been presented in the micro-ROS Kobuki demo.

ACKNOWLEDGMENT

This work has been supported by the EU-funded project OFERA (micro-ROS) under Grant No. 780785. Thanks go to eProsima for support on the setup of the Olimex board and the Kobuki example.

REFERENCES

[1] P. Bouchier, "Embedded ROS," IEE Robotics and Automation Magazine, vol. 20, no. 2, pp. 17-19, 2011.

[2] H. Tasaki, M. Tomoya, T. Kazuyoshi and T. Naofumi, "mROS: A Lightweight Runtime Environment for Robot Software Components onto Embedded Devices," in Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies, Nagasaki, Japan, 2019.

[3] "Micro-ROS website," [Online]. Available: https://micro-ros.github.io/.

[4] D. Casini, T. Blaß, I. Lütkebohle and B. Brandenburg, "Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling," in 31st Euromicro Conference on Real-Time Systems (ECRTS 2019), 2019.

[5] "rclc package," [Online]. Available: https://github.com/ros2/rclc/.

[6] "Kobuki Demo Website," [Online]. Available: https://micro-ros.github.io/docs/tutorials/demos/kobuki_demo/.

[7] "Thin-kobuki driver," [Online]. Available: https://bitbucket.org/ggrisetti/thin_drivers/src/master/.