D2.9

Bridge logic software release
Revised Y2

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D2.9
Deliverable name	Bridge logic software release
Date	December 2019
Dissemination level	public
Workpackage and task	2.3
Author	Juan Flores Muñoz (eProsima)
Contributors	
Keywords	Hardware-Bridge, Raspberry-Pi, requirements, ROS, ROS2
Abstract	

Contents

1 Summary

On this document, we will continue with the work performed on the previous related deliverable **D2.8 Bridge logic software release - Release Initial** which include, the analysis of the software requirements of the micro-ROS hardware bridge, proposition of auxiliary tools to help on the software implementation task and finalizing with some demonstration that validates the proposed approach. On this deliverable, we will expose the improvements performed on the auxiliary tools and the new features implemented on the hardware bridge software suite.

2 Acronyms and keywords

Acronym	Explanation
6LowPan	IPv6 over Low power Wireless Personal Area Networks
UART	Universal Asynchronous Receiver-Transmitter
ROS	Robot Operating System

3 Overview to Results

This document provides links to the released software and documentation for deliverable D2.9 *Bridge logic software release* .

3.1 Links to Software Repositories

The Micro-ROS bridge logic software includes a set of repository, which are list on the next list:

ros2-performace: Set of ROS 2 tools which include a cross-compilation tool to build micro-ROS Agent for Raspberry Pi 3. This is a fork of the work performed by IRobot.

- Git repository: https://github.com/micro-ROS/ros2-performance

Branch	Latest commit	ROS 2 version
microros_cc_tool	01aa75eda4be4d9276d470ff897c7248b63e95af	dashing

micro-ROS-build: Build system for micro-ROS. This branch includes the implementation of the new cross-compilation tool.

- Git repository: https://github.com/micro-ROS/micro-ros-build

Branch	Latest commit	ROS 2 version
agent_cc	cdcfe2b9e41e289ef56ca896b741c8044e3efac7	dashing

micro-ROS-bridge_RPI: Repository which includes the instructions of how to set-up the micro-ROs Hardware Bridge, the 6LowPan software configuration script and a set of examples to try the 6LowPan/NuttX interoperability.

- Git repository: https://github.com/micro-ROS/micro-ROS-bridge_RPI

Branch	Latest commit	ROS 2 version
new_bridge_tools	68303d4fb86dfaddcef9714ed4b18e023de88863	dashing

NuttX: Selected by default RTOS for microContollers.

- Git repository: https://github.com/micro-ROS/nuttx

Branch	Latest commit
micro-ros_6lowpan	cc499e835203b29a46516ec657bde859edd8aecf

4 Requeriments

The requirements for the micro-ROS hardware bridge are defined by the micro-ROS features. This requirements give this brief list of points:

- Serial communications support.
- 6LOWPAN support.
- Ethernet support.
- micro-ROS Agent full-stack support.
- ROS 2 support.
- ROS 2 and micro-ROS dependencies support such as full C++, YAML, and Python support.
- Tool to improve the set-up of the hardware bridge.

Most of the required points were achieve fully or patially on the initial release. After the initial release it left to finish the next list point, which could considerate the list of requirements for this software release:

- Achieve micro-ROS 6lowpan communication.
- Improvement and simplify the cross-compilation tool.
- Optimize the set-up process.

- Integration on the micro-ROS build system.

In the following sections we're going to show the improvements performed on each point to achieve the requirements.

4.1 Micro-ROS supports 6LowPan protocol

The 6LowPan is a communication protocol, which allows mesh networking over IPV6. The use of IPV6 addressing instead a custom addressing protocol, is an advantage in comparison with other mesh network protocols, because it allows easily the packet exchange between the 6LowPan network and the Internet. This requirement is derived from the workshop use case, which needs to use this communication protocol. On the previous micro-ROS bridge software release, was not possible to try this functionality, due to at that moment, the IPV6 support was a work in progress feature. With the latest releases of Micro-XRCE-DDS, the IPV6 addressing is fully supported, giving us the chance to try 6LowPan communications on Micro-ROS.

To check this functionality, was necessary to do the next list of modifications: - Update the version of micro-ROS on the hardware bridge, from Crystal to Dashing. This new version implements the new micro-XRCE-DDS Agent version with IPV6 support. - Create a new NuttX config profile which includes the 6LowPan configuration with the necessary micro-ROS configuration.

Due to the usage complexity of the 6LowPan stack, the release of clear and easy examples is still a work in progress point.

4.2 Cross-compilation tool

The cross-compilation tool is an auxiliary utility which improvements significantly the building of required software for the micro-ROS Hardware. This tool is required due to the building time on the real hardware could take up to ten hours.

On the first version of the cross-compilation tool, we adapted an already existing tool (ros2_raspbian_tools) to compile micro-ROS Agent, micro-ROS client and ROS 2 for Raspberry Pi 3. This tool reduced the building time to less than three hours, which is considerably improved. But after some month of usage and the feedback received, we think that it is only necessary to add micro-ROS Agent because the rest might be useless for the hardware bridge, which should only act as a communication bridge between the micro-ROS and the ROS 2 world. On the other hand, at the time that we try to update to Dashing version, we notice the high difficulty to maintain and update this suite of software, so we thought that it was necessary to improve it and develop or find a much simpler tool.

The first considered option was the official ROS 2 supported tool: - AWS ROS 2 Tooling Cross-Compile

This tool realizes the building process on a processor emulator called Qemu. This specific implementation was configured to emulate the CPU of the Raspberry Pi 3 which is an **ARMhfv7**'. This is an interesting approach because it simplifies the cross-compilation process, giving a like experiencing real hardware. But from the other side, the building time is similar or even worse than on the real hardware, giving as conclusion the rejection of this tool because it doesn't improve the building time problem with the real hardware.

Continue with the research of a new and better tool which fits our requirements we found a cross-compilation tool developed by IRobot. This utility is part of a set of ROS 2 performance benchmarking tools. After trying it, we notice a significantly improve on compilation time of ROS 2 for Raspberry Pi, this improvement translates into less of one hour of building time, the first execute time and around 30 minutes on afterward test.

With these results, it could achieve our first requirement which improved the building time, but we're going to analyze the internal structure to decide if it improves our next requirement, which is easy to maintain. This tool performs the next point to achieve a cross-compilation of ROS 2:

- Create a docker base on Ubuntu 18.04 and install the necessary tools to download and set-up ROS 2 Dashing.
- Run a Raspbian Docker with ARMV7HF architecture base on a Qemu emulation to obtain the Raspbian file system with all the necessary libraries for ROS 2, compiled for ARM architecture.
- The cross-compiled libraries and the source code of ROS2 is copied to the first Ubuntu docker.
- On the Ubuntu docker is compiled ROS 2, using a toolchain that includes the ARM compiler and linking the cross-compiled libraries.
- When is compiled, it's exported the ROS 2 workspace from the docker to the host PC.

If we execute on subsequent times with newer version or modifications of ROS 2, is not necessary to recompile everything, it's only necessary to recompile the source code of ROS 2, reducing, even more, the compilation time. To adapt to our requirements, as a first step it was only necessary to modify the source code that we want to build, from the ROS 2 source code to the micro-ROS Agent minimum repos. To optimize the tool and fully adapt to the micro-ROS project, we did the next modifications: - Delete the ROS 2 version selection. - Delete the installation of unnecessary libraries on Raspbian, to improve build time - Delete the final copy step of the compiled code. - Cosmetic modifications to adapt to the micro-ROS naming. - Modification on the file permissions to avoid the root usage on the final result. This improves the compatibility with continues integration tools. - Add to the compilation micro-ROS Agent.

4.3 Integration on the build system

To unify and simplify the micro-ROS usage experience and the other hand to unify the building process with ROS 2, it was developed the micro-ROS build system. This tool allows to build the different part of micro-ROS a like a ROS 2 package.

We have created a new option on the build system to set-up all the required software for the micro-ROS hardware bridge. This integration on the build system, download the micro-ROS cross-compiler tool, execute the utility to obtain the cross-compiled code and finally copy the build result and the 6LowPan configuration script to a folder that clarifies to the user that is necessary to copy to the Raspberry Pi file system.

5 Tutorials

In our effort to provide quality documentation, we have to rewrite the Hardware bridge set-up document, which can be found here: Micro-ROS Hardware Bridge Set-Up

With these modifications, we aim to unify on the same document the 6LowPan configuration and the micro-ROS Agent set-up giving a related vision between both tutorials.

6 Future steps

From now our efforts will be focus on maintaining this suite of software and update to the latest version of micro-ROS. Also one of the weak points of the bridge logic set-up is that for inexperienced users, it can be a complex task, so we should try to simplify as much as possible the usage of these tools, providing a simpler auxiliary tool.