D3.12

Interoperability Report Y3

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D3.12
Deliverable name	Interoperability Report
Date	June 2020
Dissemination level	Public
Workpackage and task	WP3 - Task 3.3
Author	Borja Outerelo Gamarra (eProsima)
Contributors	Pablo Garrido Sánchez (eProsima), José Antonio Moral Parras (eProsima)
Keywords	Micro-ROS, Report, Robotics, ROS, microcontrollers, middleware, Interoperability, ROS2, FIWARE, SOSS
Abstract	This document provides a report regarding the interoperability of micro-ROS with existing ROS2 and with FIWARE.

Contents

1 Acronyms

Acronym	Explanation
CDR	Common Data Representation
DDS	Data Distribution Service
DDS-XRCE	DDS For Extremely Resource Constrained Environments
GA	Grant Agreement
LTS	Long Term Support
OFERA	Open Framework for Embedded Robotic Applications
ROS	The Robot Operating System
RTPS	Real Time Publish Subscribe
SOSS	System Of Systems Synthesizer
WP	Work package

2 Executive summary

This report, **D3.11 Interoperability Report - Y2**, provides an overview of the status of the interoperability of micro-ROS with other protocols, on the third year of the project.

OFERA consortium focused on two micro-ROS interoperability points during the reporting period:

- Interoperability with ROS 2 and ROS systems.
- Interoperability with FIWARE via SOSS.

Regarding ROS 2 and ROS system interoperability, OFERA consortium has been adopting more ROS 2 concepts and updating existing developments to be aligned and interoperable with the new ROS 2 releases, including the LTS releases of ROS 2. Maintaining the initial architecture approach allows keeping micro-ROS and ROS 2 interoperable at different levels.

In micro-ROS' architecture definition, the interoperability with the current ROS 2 middleware protocol, DDS/RTPS, was taken into account. The middleware protocol that the consortium chose for micro-ROS, DDS-XRCE, provides this DDS interoperability.

The Object Management Group (OMG) created the DDS-XRCE protocol with this interoperability as one of its primary objectives. DDS-XRCE grants interoperability between its clients and the existing DDS global data spaces. DDS-XRCE achieves this interoperability thanks to the use of a DDS-XRCE Agent which acts as a bridge between DDS-XRCE networks and a DDS global data space.

Taking all of the above into account, using DDS-XRCE as a middleware and granting DDS interoperability allows the interoperability between micro-ROS and ROS 2 systems, in the current status of ROS 2 implementations.

The interoperability with ROS 1, in the same way as last year, can come from two different sources, either: 1) using the existing ROS 2-ROS 1 bridge or 2) using SOSS with the ROS 2 and the ROS System Handles. SOSS is a tool developed by EPROS and OSRF that allows bridging different protocols, and thanks to the ROS 2 System Handle, it is the new standard way to interoperate with ROS 2.

Besides from providing communication with the ROS 2 data space, SOSS allows to communicate with any other middleware whose System Handle implementation is available. Among them are, for example, FIWARE Context Broker, ROS 1 and ROS 2 System Handles, which means that communication between micro-ROS and ROS 1, ROS 2 or FIWARE applications can be addressed seamlessly.

Also, an effort was made to migrate the SOSS platform to a common Dynamic Data representation, using OMG's DDS-XTypes specification, which takes advantage of the IDL language as a means for easily describing the types used in an exchange of information among applications. This was adopted thanks to eProsima's *xtypes* implementation.

Further documentation about SOSS, as well as the supported middlewares up-to-date, can be found here.

micro-ROS' interoperability with ROS2 and ROS was achieved within the framework defined by WP3 and more precisely *task 3.3: ROS bridging & interoperability*.

micro-ROS' interoperability with FIWARE was achieved within the objectives set by WP3, involving in particular *task 3.4: Fiware interoperability* achieved micro-ROS' interoperability with FIWARE.

3 Introduction

3.1 Purpose of document

The purpose of this document is to provide an overview of the interoperability of the Horizon 2020 Innovation Action shortly named OFERA - Grant Agreement Number 780785 of the European Commission. It covers the third period of the project, from 31st December 2019 to 30st June 2020 (six months, from M24 to M30). This interoperability report targets two different systems and their micro-ROS interoperations: ROS 2 (ROS), and FIWARE. Hardware interoperation, depending on HRIM, has been dropped from this report as Acutronic left the project and the partner taking over most of their tasks, eProsima, at the moment has not any development plans on the existing HRIM work.

This report follows this structure:

- Part 1 is the executive summary with a quick review of the advances achieved.
- Part 2 is the present introduction.
- Part 3 is the report itself. It:
  - Explains ROS 2 interoperability
  - Explains FIWARE interoperability
- Part 4 contains conclusions.

This document is a revised version of last years' periodic reports:

- Interoperability Report Y1 - 31.12.2018
- Interoperability Report Y2 - 31.12.2019
- Interoperability Report Y3 - 30.06.2020

4 Interoperability report

In this section, the report defines and explains the interoperability with ROS 2 and FIWARE, regarding their status halfway through this third year of OFERA project.

4.1 ROS 2 Interoperability

4.1.1 Introduction

micro-ROS' interoperability with ROS2 was one of the technical objectives of micro-ROS as mentioned in the OFERA GA. Also, it could be seen as the must-have interoperability as it accomplishes one of the foundational ideas behind micro-ROS, bringing ROS 2 to embedded devices.

ROS 2 interoperability and portability has been a critical piece on the design and development of the current micro-ROS version. The current version of micro-ROS pursues to resemble as much as possible the usage of ROS 2. For that reason, micro-ROS uses the same concepts as ROS 2, and even it reuses a significant amount of the tools surrounding the ROS 2 environment.

This reuse of existing concepts and tools allows micro-ROS to achieve a high level of integration with current ROS 2 systems. At the stage captured by this revision, apart from continuously using and evolving the micro-ROS architecture approach, micro-ROS reuses ROS 2 build system and types definitions. Below we expose the two features that ease, to a great extent, the interoperability between micro-ROS and ROS 2:

a. The reuse of the ROS 2 build system allows having the same process for generating interfaces and their associated and required libraries. In this direction, a new micro-ROS-build package has been created to gather all micro-ROS build configuration in only one package. With this new package, users can build the micro-ROS-Agent, micro-ROS demos and cross-compile micro-ROS applications for the different supported platforms within a ROS 2 environment. This way, working within a micro-ROS workspace does not differ from regular work with ROS 2 workspaces.
b. ROS 2 provides different ways of defining types used in its middleware user applications, either by means of *msg/srv* files (for messages and services, respectively) or *IDL* type definition files. These types are used for generating code used by ROS 2 to recognize them and operate with them (briefed as the *TypeSupport*). Reusing these types definitions in micro-ROS as the base for its specific code generation and type support needs ensures that mismatch issues are reduced as much as possible and assures compatibility with ROS 2 types definitions.

Great effort is being made in keeping this ROS 2/micro-ROS compatibility up-to-date. For example, during this year, a new version of ROS 2 has been released, Foxy Fitzroy, which is a LTS release of ROS 2. This means that the upcoming micro-ROS release, based on ROS 2 Foxy, will be long term supported as well.

Apart from those high-level integrations and message definitions, micro-ROS is designed to use standard middleware protocols, natively interoperable with the one used in ROS 2: DDS.

The protocol chosen for enforcing communication and interoperability with the DDS world (and, hence, the ROS 2 data space) is OMG's DDS-XRCE protocol. This protocol comes from the same

standarisation body as DDS, and was specifically designed to address the task of communicating resource constrained systems with the DDS data space, keeping interoperability as a foundational requirement. Therefore, micro-ROS can be interoperable with any DDS vendor. Taking into account the nature of DDS-XRCE protocol structure, clients (executed in resource constrained devices) and agents (which act as a bridge between micro-ROS and the rest of the middlewares, via DDS protocol) can interoperate between them, even if their implementations are provided by different vendors, given that DDS-XRCE standardizes the communication mechanism.

Interoperation between several vendors' implementations of the DDS standard is out of the scope of this project. However, it is worthwhile emphasizing that OMG standardisation was designed with interoperability in mind, providing multiple levels of interoperation, ranging from the DDS interface to the interoperable wire protocol used (RTPS), the data representation (IDL) and serialization mechanism (CDR). DDS vendors and users continuously test this DDS vendor interoperability in different ways; two of them are worthy of being mentioned here:

1. OMG official GitHub repository [1] with interoperability tests. DDS vendors manually run these tests.
2. ROS 2 official CI site with automated tests [2], using different rmw implementations. The OSRF runs those tests on a CI tool, Jenkins. Sample results of an execution: Jenkins test run.

The use of such standardised and DDS interoperable protocols, as this report explains later, eases the interoperability between ROS 2 and micro-ROS at the middleware layer.

4.1.2 micro-ROS current status

As introduced above, OFERA achieved micro-ROS and ROS 2 interoperability from the early stages of the project. This interoperability was eased thanks to the use of interoperable middleware protocols, DDS and DDS-XRCE on ROS 2 and micro-ROS sides respectively.

As already mentioned, the current ROS 2 implementation uses OMG DDS as the underlying middleware protocol. Using DDS as a middleware allows any DDS implementation to be interoperable, almost out of the box, with ROS 2. eProsima, the Consortium coordinator, provides one of the DDS implementations, Fast DDS, which has been chosen to be the ROS 2 default middleware implementation.

The eProsima's situation, being the ROS 2 default middleware provider and an OMG member participating in the standardisation of these protocols, provides micro-ROS with a good advantage position. eProsima, apart from the Fast DDS implementation of the DDS standars, also develops and maintains the current micro-ROS middleware implementation, Micro XRCE-DDS, which implements the OMG's DDS-XRCE protocol. Being implementer of both ends in the interoperability, Fast DDS and Micro XRCE-DDS, ensures that both products will keep interoperable from early development stages to the latest standard modifications. Both the users and eProsima proved this interoperability in different ways:

a. eProsima uses CI with integration tests [3]. These tests use Micro XRCE-DDS Agents and Clients, along with Fast DDS, testing communication between all the pieces.
b. Third parties: Renesas and Robotis are actively using micro XRCE-DDS. They mainly use Micro XRCE-DDS to communicate with ROS 2 (DDS).

- Renesas enables its platform GR-ROSE to be a ROS 2 participant using Micro XRCE-DDS [4].
- Robotis uses Micro XRCE-DDS as a bridge to communicate their XEL Network with ROS 2 [5].

During this 6-months-long reporting period, Micro XRCE-DDS has been ported and adapted to the new Fast DDS release, 2.0, which is the core for the Foxy Fitzroy ROS 2 release. Adapting Micro XRCE-DDS Agent to the use of this new Fast DDS versions ensures middleware interoperability with ROS 2 different versions.

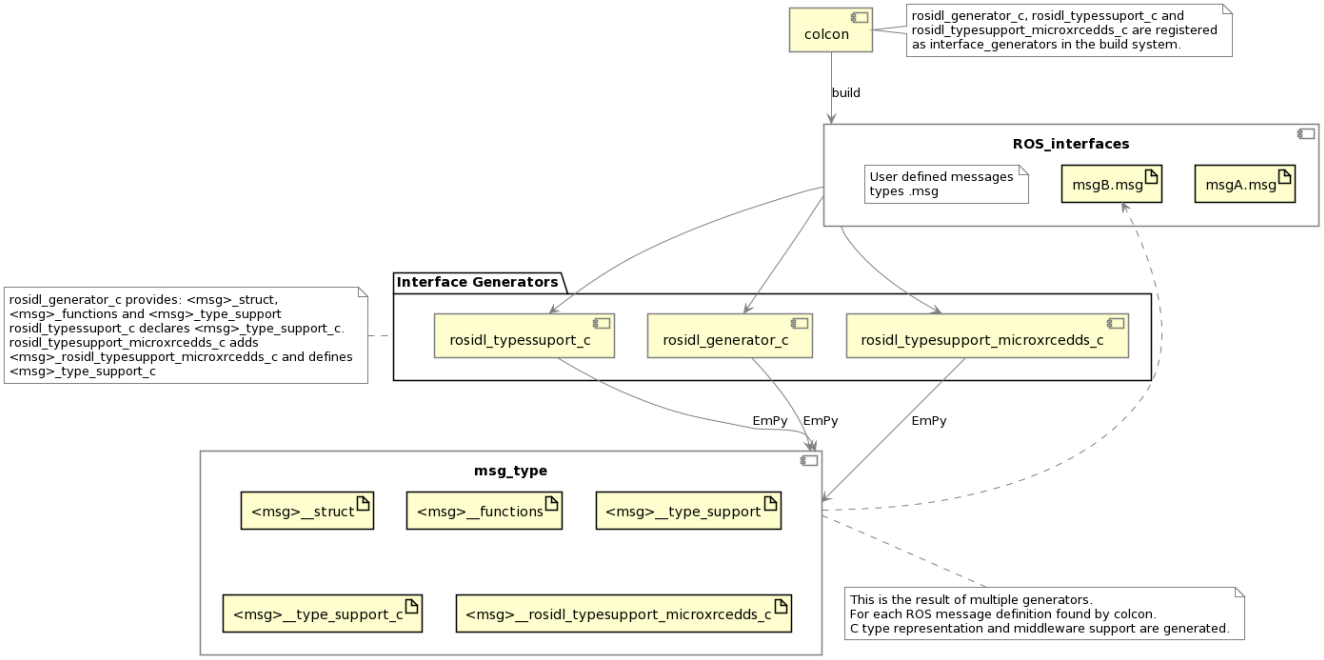
Even though this middleware interoperability is the base to a ROS 2 - micro-ROS interoperability, it is not the only interoperability needed for achieving full integration. One of the critical aspects of ROS 2 is the standardization of their interfaces, that is, simplifying, the interoperability between applications. ROS 2 interface plays a prominent role in this task. As an addition to last years' publish-subscribe interface, services, actions and parameters have been supported as part of the work of this year. Services, actions and parameters have been implemented in the upper layers of micro-ROS, on top of the RPC mechanisms introduced in the DDS-XRCE standard in combination with the publish-subscribe interfaces.

During this year, micro-ROS has been adapted to support the new IDL format supported by ROS 2 build-time tools. micro-ROS currently partly supports the types and same type definition as ROS 2 does. There is support for all the basic types and some of the vectorised ones. This new type support was added as part of a new release supporting LTS version of ROS 2 Dashing Diademata. micro-ROS does not support all the complex type vectors as could be arrays of strings and similar "complex types".

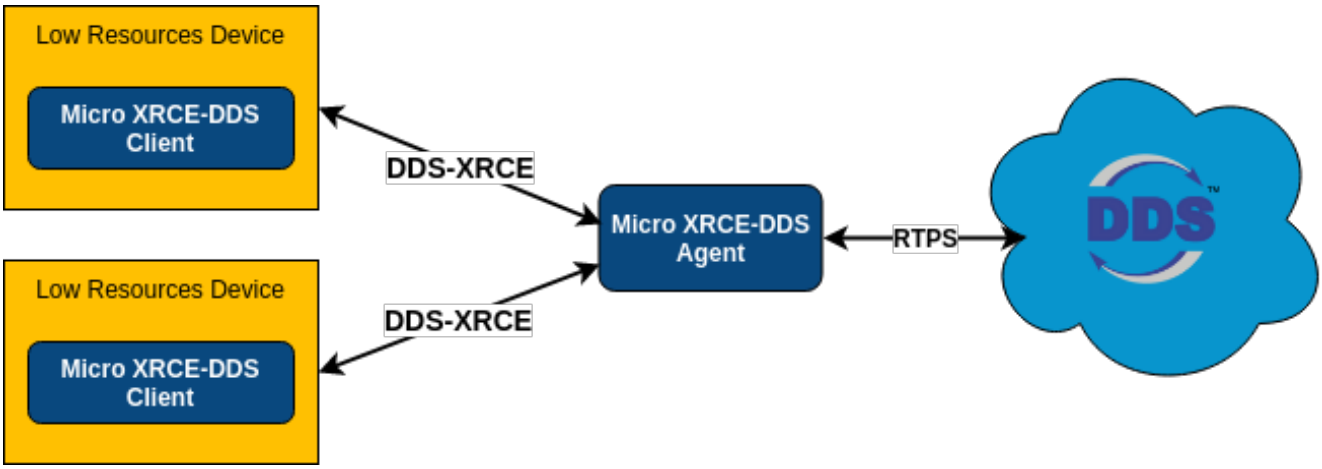
OFERA has provided demonstrations of ROS 2 messages from and to micro-ROS systems in the current micro-ROS version. The consortium keeps updating these demonstrations in micro-ROS-demos. These demonstrations use the mechanism mentioned above where automatic generation of type support is achieved on both ends, ROS 2 and micro-ROS, using the same source type definition file. These demonstrations show the users how to include new types and how straightforward it would be to integrate any types in both ends of the communication, in this case, micro-ROS and ROS2.

4.1.3 Architecture

ROS 2 interoperation is directly related to type support and middleware implementations. In the case of micro-ROS, consortium members have developed a type support mechanism similar to the one used in ROS 2. In micro-ROS, the build system generates type support libraries in C language for each interface found in the workspace. The source of the generation is precisely the same as in ROS 2, what allows the reuse of types definition by easing the port of types from one end to the other end of the interoperable system. The following figure shows this interface generation process.



As stated above, another key point to allow micro-ROS - ROS 2 interoperability is the usage of interoperable middleware. On the ROS 2 end, the middleware used is based on DDS. On the other side, micro-ROS has been conceived and developed using a new OMG standard 100% interoperable with DDS. This new standard is DDS-XRCE, and allows to communicate embedded devices with DDS networks natively. DDS-XRCE provides the interoperability between micro-ROS devices with existing ROS 2 applications at the middleware level.



This reuse of type definition, together with the usage of interoperable middleware, is what provides micro-ROS with native interoperation with ROS 2. This interoperability ebodies an entry point for a wide range of interoperations for micro-ROS, for example that with ROS 1, allowed by the existing bridge from ROS 2 to ROS 1, which will ensure that micro-ROS can in turn interoperate with ROS 1 systems hopping over ROS 2.

Also, one of the most significant changes regarding the communication between micro-ROS applications and other middlewares is provided by the development of SOSS, during the past and present year. Although it is explained in detail later on in this document, it could be summarized as an inte-

gration tool for communicating any two middlewares of different nature. As we'll see, this implies that SOSS can be a powerful tool for communicating micro-ROS data spaces with other systems besides those based on ROS 2.

4.2 FIWARE Interoperability (FIWARE System Handler and SOSS)

4.2.1 Introduction

Regarding FIWARE interoperability with micro-ROS, a new architectural approach was addressed last year. Indeed, during this year, a new integration tool called System of Systems Synthesizer (SOSS), has been developed and released. SOSS provides a set of tools and configuration utilities to interconnect different communication middlewares. SOSS is based on a plugin system (with the plugins named System Handles) and a common data representation allowing any middleware endowed with a System Handle to communicate with SOSS.

Crucially, dedicated System Handles have been developed for both FIWARE and ROS 2, so that proven interoperability exists between these two systems. As a consequence, micro-ROS interoperability with FIWARE is done using SOSS utilities and taking advantage of the micro-ROS - ROS 2 interoperability detailed above. The FIWARE - micro-ROS interoperability is thus implemented via ROS 2 (DDS), the same way as the previous ROS interoperability, achieved by hopping over ROS 2.

The OSRF plans to adopt SOSS as the default ROS 2 integration mechanism, since it provudes a well-defined way to connect ROS 2 with any other system. eProsima's Integration Service is a tool based on SOSS that allows any DDS-based system to communicate with any other protocol. Given that the micro-ROS Agent is able to understand information from the DDS world, Integration Service can be used to communicate a micro-ROS application with any other middleware-based application. Also, a ROS-Industrial FTP outcome: the initial release of ROS2 Integration Service (ROSIN) uses SOSS (System Of Systems Synthesizer) as core communications enabler.

The bigger change uexperienced by the FIWARE interoperability during this last year regards a major change undergone by SOSS, both in its core and in the System Handles allowing it to communicate with all the middlewares supported. This change affects the common data representation used by SOSWS, which passed from a non-standardized SOSS-specific representation to the standardized OMG-defined eXtensible and Dynamic Topic Types for DDS (DDS-XTYPES).

Therefore, while during the second year of the OFERA project the interoperability was based on this initial version of SOSS, as of the third year of project it has been changed to use this new common data representation both in the SOSS core and in the FIWARE and ROS 2 plugins, or System Handles.

4.2.2 micro-ROS current status

Using SOSS and thanks to the micro-ROS - ROS 2 interoperability micro-ROS is able to interoperate, not only with FIWARE but with any other system capable of being integrated with SOSS. Currently SOSS supports: ROS 1, ROS 2, DDS, Websocket and FIWARE, with more integrations under development.

The micro-ROS - FIWARE integration was achieved using SOSS and two of its System Handles, ROS 2 and FIWARE. In this time and thanks to SOSS, the integration was easier compared to last year as each System Handle has the specific details of its parent protocol, NGSIv2 in the case of FIWARE

and DDS topics in case of ROS 2. This abstraction of the other end of the communication is one of the main reasons that makes the development of SOSS a very interesting challenge, because, from now on, adding further integrations and interoperability capabilities between micro-ROS and other middlewares will be easier.

4.2.3 Architecture

SOSS is based on a plugin system; each one of these plugins is called a *System Handle*, and provides everything necessary to publish/subscribe to/from topics of a specific middleware, as well as to perform request/reply petitions with a server/client based architecture.

This tool's functioning begins with a first step in which the user configures the behaviour of the communications between two or more middlewares by means of a YAML file. In this YAML file the communication channels between System Handles are defined, as well as the types of the data which are going to be exchanged. Typically, these types will be defined using OMG'S IDL language definition. During a second stage of the communication implementation, OMG's DDS-XTYPES have a key role as the standard language that each System Handle is able to speak, providing both these plugins and the core with a common data representation based on the IDL definition specified in the configuration stage of each specific SOSS instance.

The SOSS architecture can be summarised with the following diagram:

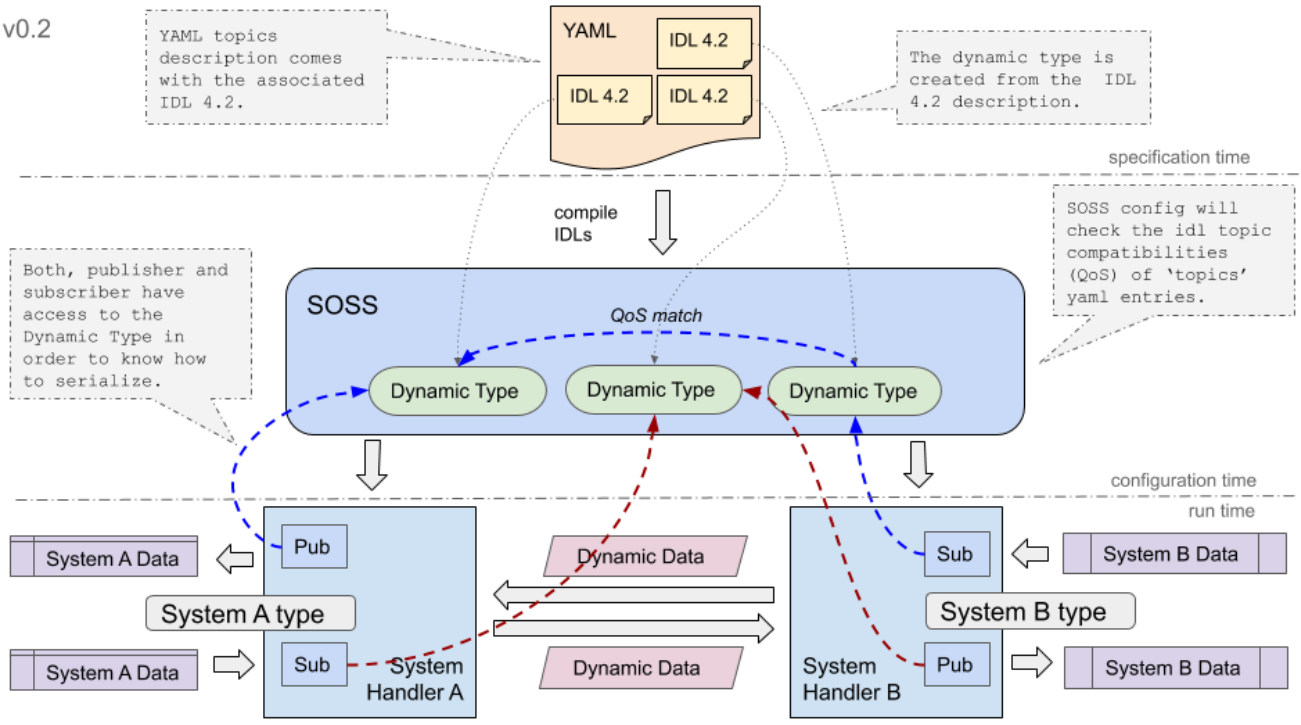


Figure 1: SOSS full architecture

SOSS has been developed with the idea to be integrated not only with new ROS 2 installations, but also with existing ROS 2 ones. For that, new type support has been developed in which SOSS

generates dynamic types based on existing ROS 2 types. In the following diagram, this mechanism and the files generated are detailed.

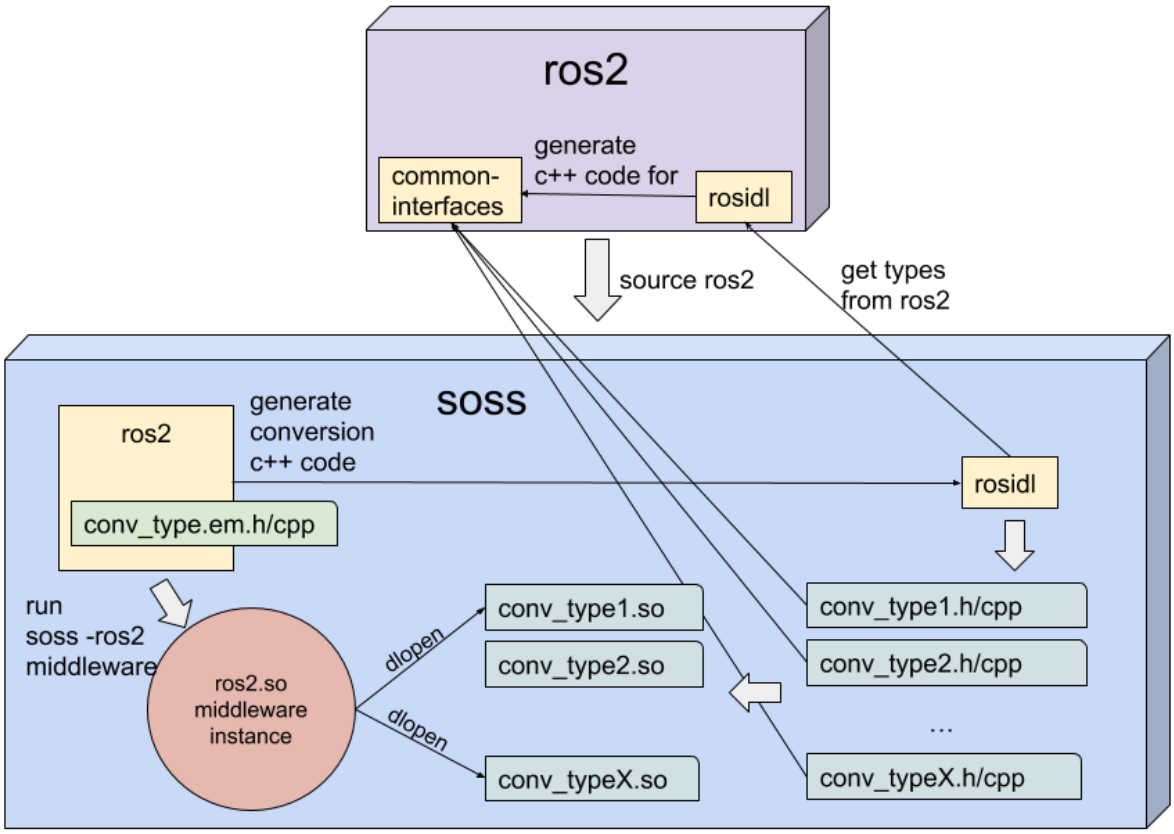


Figure 2: SOSS type support

micro-ROS' integration with FIWARE gets the advantage of this new system. The micro-ROS - FIWARE Context Broker integration is done using SOSS along with the ROS 2 and FIWARE System Handles. micro-ROS has a "native" integration with ROS 2. In this case, micro-ROS communicates with ROS 2 without the need of any System Handle or SOSS intervention. Then, thanks to the use of SOSS and the FIWARE System Handle, ROS 2 can publish and subscribe to FIWARE Context Broker data. As micro-ROS subscriptions and publications from outside are seen as regular ROS 2 subscriptions and publications, the FIWARE System Handle can make use of them out of the box.

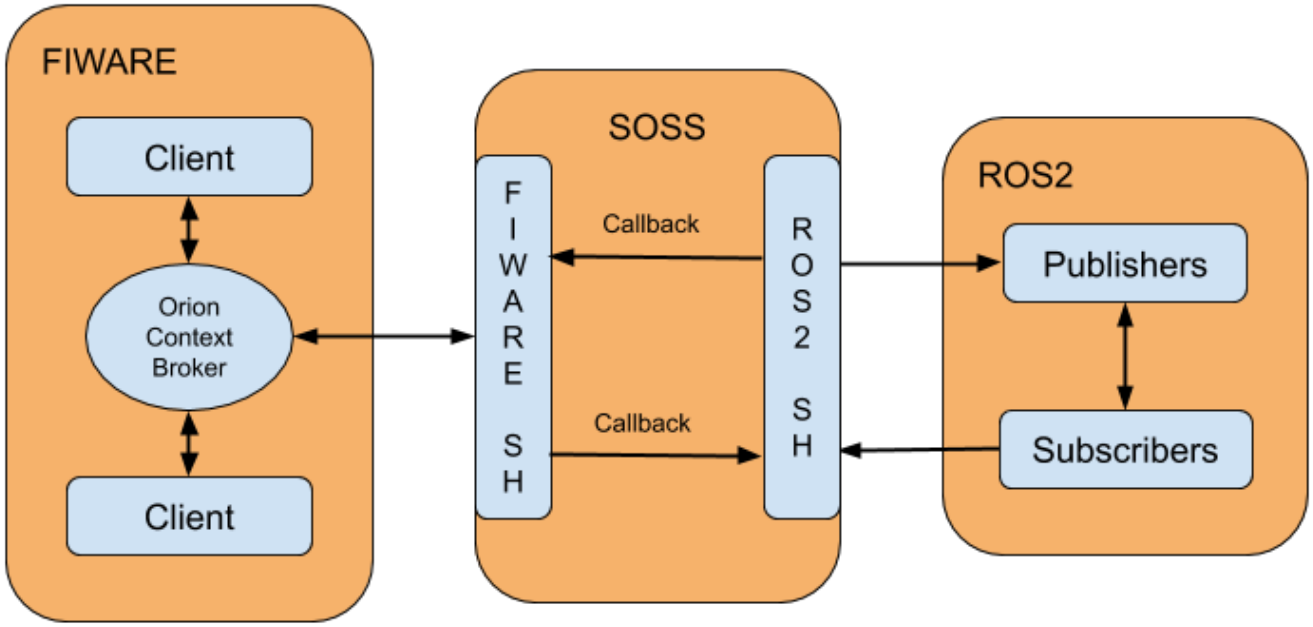


Figure 3: SOSS ROS 2 - FIWARE

Also, a new remapping feature has been introduced in SOSS, taking the capabilities of this tool even further. Remapping offers the possibility of renaming the topic name and topic type of a specific communication channel, defined in SOSS by means of the YAML configuration file, for every one of the involved System Handles, what allows avoiding the tedious task of renaming topics and types for applications which are already developed, and making the integration process much more configurable and powerful.

4.3 Conclusion

During the reporting period, ROS 2 interoperability has been improved, from the types and interfaces perspective. FIWARE interoperability has been reformulated around a new software release.

Both ROS2 and FIWARE interoperability were achieved relatively easy thanks to the use of standard protocols (DDS and DDS-XRCE). One of the most time-consuming efforts was aligning version with ROS 2 releases, but with the already existing support of ROS 2 LTS version achieved this year, the work for future releases will be easier.

ROS 2 integration needs further development to support all the data types. However, developing services was a big step towards full interoperability. In the future, we plan to develop interfaces, more precisely, parameters and events. These are two ROS 2 concepts that will ensure better interoperability between micro-ROS and current ROS 2 systems.

From the FIWARE point of view, interoperability has been achieved with the same extension as the one between ROS 2 and FIWARE. Future native micro-ROS interoperability could be achieved developing a micro-ROS System Handle which will allow micro-ROS to communicate not only with FIWARE but also with any other System integrated into SOSS.

The following steps in this sense will be to develop more ROS 2 concepts and a micro-ROS System Handle for SOSS integration.

References

[1] O. (Object Management Group), 'Interoperability examples for the data-distribution service (dds) standard'. 2018 [Online]. Available: https://github.com/omg-dds

[2] O. (Open Source Robotics Foundation), 'Official ros 2 communication tests'. 2018 [Online]. Available: https://github.com/ros2/system_tests/tree/master/test_communication

[3] eProsima, 'Micro xrce-dds integration tests'. 2018 [Online]. Available: https://github.com/eProsima/Micro-XRCE-DDS

[4] godzilla-max, 'A dds-xrce implementation for rx65n'. 2018 [Online]. Available: https://github.com/godzilla-max/rose_sketch

[5] Robotis, 'Robotis xel network integration with micro xrce-dds'. 2018 [Online]. Available: https://xelnetwork.readthedocs.io/en/latest/