# D7.14

# Long-term maintenance and evaluation plan
# Revised Y4

| | |
|---|---|
| Grant agreement no. | 780785 |
| Project acronym | OFERA |
| Project full title | Open Framework for Embedded Robot Applications |
| Deliverable number | D7.14 |
| Deliverable name | Long-term maintenance and evaluation plan |
| Date | December 2021 |
| Dissemination level | public |
| Workpackage and task | 7.4 |
| Author | Francesca Finocchiaro (eProsima) |
| Contributors | Ralph Lange (Bosch), Ingo Lütkebohle (Bosch), Borja Outerelo Gamarra (eProsima), Iñigo Muguruza Goenaga (Acutronic Robotics) |
| Keywords | micro-ROS, robotics, ROS, microcontrollers |
| Abstract | Definitive plan for long-term maintenance and evolution of micro-ROS after the end of the project. |

# Contents

# 1 Acronyms and keywords

| Term | Definition |
|---|---|
| **ROS** | Robot Operating System |
| **OSRF** | Open Source Robotics Foundation |
| **MCU** | Microcontroller Unit |
| **CPU** | Central Processing Unit |
| **RTOS** | Real-time Operating System |
| **DDS** | Data Distribution Service |
| **XRCE-DDS** | Extremely Resource Constrained Environments DDS |
| **CI** | Continuous Integration |
| **RCL** | ROS Client Library |
| **RMW** | ROS Middleware Interface |
| **RCLC** | ROS Client Library for C language |
| **RCLCPP** | ROS Client Library for Cpp language |
| **TSC** | Technical Steering Committee |
| **LTS** | Long Term Support |
| **WG** | Working Group |
| **SIG** | Special Interest Group |
| **AL** | Abstraction Layer |

# 2 Intro

In this document, we will introduce the OFERA long-term maintenance and evolution plan of the relevant project results. To provide context, the OFERA project aims to bring resource-constrained devices, such as microcontrollers, as first-class participants in the robot ecosystem. In particular, it aims to offer support to the Robot Operating System (ROS) on its second version: ROS 2. The document will discuss the current strategy towards making the results accepted, maintained and further developed, by extending what previously stated in the document *D7.13 - Long-term maintenance and evaluation plan - Initial*.

# 3 Structure

First of all, we provide background information about ROS, where we describe the development status, how the software is organized and the new governance structures that have been created by the Open Source Robotics Foundation (OSRF). Secondly, the possible accommodation of micro-ROS software entities in ROS 2 source-code is depicted, where the similarities and the divergences are explained, package by package. We also present the candidates which we want to be in charge of each package. Thirdly, the conclusion is presented.

# 4 Background on the ROS 2 development model

This project targets ROS 2, which already integrates basic requirements for real-time, industrial and embedded-system use (see why ROS2) and thus provides a sufficiently mature base.

ROS 2, like ROS before it, is organized into modules called "packages", which are the basic unit of dependency management. Active packages are looked after by a developer, called "maintainer". This maintainer could be a member of the Open Robotics, a company developer or an individual that overviews, releases and manages the package. ROS provides a maintenance guide about the role the maintainer must perform.

Most importantly for the purposes of this document, when code is *contributed* to a package, the maintainer is responsible for checking it and, if accepted, is *thereafter expected to maintain it*. This transfers the maintenance burden and is therefore an attractive strategy to minimize future work, but of course, this approach has practical limits. When making large contributions, their continued maintenance cannot always be off-loaded to the existing maintainer, but the contributor is expected to shoulder some of this burden.

Temporally, the development and maintenance of ROS is organized into so-called distributions, where "a distribution is a versioned set of ROS packages". Within a distribution, API changes to the core are avoided, but when changing distributions, such changes may occur. Such API changes are currently more significant in ROS 2, as it is in its early stages, feature and performance discussions are taking place.

Distributions are normally released every 6 months, and in principle, once a distribution is released, older distributions need no longer be maintained, except for so-called Long-Term-Support (LTS) distributions, which have a lifetime of (currently) 2 years. In practice, packages should also be maintained on non-LTS distributions for a grace period to allow switch-over. This period has traditionally been anywhere from a few weeks up to several months or even years. For ROS2, it has so far been common practice to drop support for older distributions immediately, but since the first LTS distribution ("Dashing Diademata") has just been released, this will now change.

Moreover, each distribution is typically released for several underlying OS platforms (currently Linux, OSX and Windows) and architectures (x86 and ARM, 32bit and 64bit).

Apart from bug fixes, the change from one distribution to another is the most significant source of maintenance effort. Moreover, changes to the underlying OS platforms during the lifetime of a distribution can also cause maintenance tasks, e.g. when a dependency has been updated.

## 4.1 ROS TSC and Working Groups

One of the novelties that ROS 2 brings is the creation of the TSC. The TSC is the committee that "is responsible for the technical direction the project takes" and is "made up of individuals who contribute materially to the project and/or individuals who represents organizations that contribute materially to the project" (taken from this document). In addition, Working Groups have been established, which aims to discuss and develop ROS 2 in specific areas, such as safety-critical, navigation, security or quality assurance. The consortium members have been involved in the creation and leadership of the Embedded Working Group (EWG), which discusses the use of microcontrollers in ROS 2. This Working Group is in charge of the embedded portion of the ROS

and ROS 2 ecosystem, and organizes four-weekly meetings (EWG meetings) to gather the embedded ROS community around topics of general interest. Each time a detailed Agenda is released around one week before the meeting takes place, and community members external to the consortium are warmly invited to participate and bring to the meeting their own micro-ROS related issues. The purpose of this WG is twofold: on the one hand it aims at generating a strong community that contributes to the evolution and development of micro-ROS, by implementing desirable software components, to port micro-ROS to new platforms, and to signal any possible bug or malfunctioning in the code. On the other hand, we wish for these users to involve increasingly in the micro-ROS related tasks, with the ultimate goal of generating a self-sustainable environment in which part of the long-term support to the libraries is outsourced as much as possible.

Aside the OFERA project, some consortium companies -specifically, BOSCH and eProsima- are members of the TSC. In this TSC, they contribute to ROS 2 development performing different tasks, sponsoring ROS 2 developers and contributing with discussions and implementation of new features in ROS 2 from BOSCH side, and adapting DDS implementation for ROS 2 use from eProsima side (Fast DDS). Furthermore, BOSCH is also involved in the Real-time Working Group, where real-time specific topics are discussed. This shows the commitment the consortium members have regarding ROS 2.

Once we have overviewed the purpose of ROS 2 and its general characteristics, we will proceed to show the long-term maintenance plan for micro-ROS.

# 5 Overall Strategy

Coming back to the proposal, OFERA is a three-year project (+1 year extension) with the aim of enabling companies to create cheaper robot components using state-of-the-art technology. If there is not a long-maintenance strategy, the use of the created assets and its support could be discontinued, wasting the development efforts incurred. Notice that along its third year of life, it has been decided that the project would last one year more, thus becoming a four-years project. The last year of project (2021) will be mainly devoted to working on a long-term strategy that ensures long-term maintenance to the library on behalf of the community.

Maintenance normally is a task that is overlooked by organizations in projects. It requires a proper planning and resource dedication, which normally implies a big effort. Same applies to the outcome OFERA project is developing. The consortium is already creating a considerable amount of software distributed among different repositories. These repositories contain core elements, such as middleware or client-library implementation, and other elements, such as benchmarking tools. The required expertise to maintain all this outcome is considerable.

Taking the explained context into account, the best option for not wasting the consortium's effort in developing micro-ROS is to merge the micro-ROS repositories as ROS 2 core repositories. This will imply that the development effort would not be solely of the consortium companies, giving the chance to other companies and foundations to use, analyze or improve it. This scenario would allow having access to more resources that nowadays the project has, for example, in terms of man-power or the quality of the technical discussion happening, as there would be a larger amount of people taking care of the packages micro-ROS is composed of.

The relation of the source-code OFERA project is creating regarding ROS 2 is various, depending how the contributions fit in the actual code. We can split them into the following three kinds:

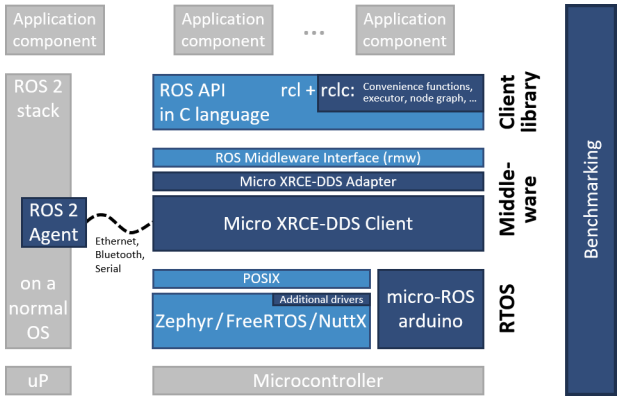


Figure 1: micro-ROS stack

- Modifications to ROS2 to improve performance/resource use.
- Added features within the core libraries.
- Supporting software specifically for micro-controllers.

The next section analyzes each point in depth. But before, mention that this source-code's elements are scattered through all the micro-ROS layers, which are depicted in the next image:

Regarding this, during the last trimester of 2020 we have undertaken the task of generating Pull Requests to add the fundamental micro-ROS repositories to the REP-2005 of ROS 2. This REP describes the common packages of ROS 2. This list is curated to include packages that are:

- open source (not proprietary);
- connected to the ROS 2 project (not general purpose software, except when included with ROS 2-specific bindings);
- broadly usable (not narrowly tailored for uncommon use cases); and
- evidently used by a nontrivial part of the community.

The aim is that all of these packages are also:

- actively maintained and following accepted software development processes, including code review and testing; and
- at Quality Level 3 or better, as defined in REP 2004; and
- are maintained by an organization or at least two individuals.

The list should be self-contained and exhaustive: if a package appears on the list, then all of its ROS 2 dependencies must also appear on the list (third-party or system dependencies should not be included).

- Micro-XRCE-DDS-Client
- Micro-CDR
- Micro-XRCE-DDS-Agent
- Fast-DDS
- Micro-CDR

- rclc
- rosidl_typesupport_microxrcedds
- rcutils
- rcl
- rmw_microxrcedds

In this same line, during 2021, we have undertaken the following:

- To assess the Quality Level of each micro-ROS related package, as as defined in REP-2004;
- To make direct Pull Requests so as to merge our changes to the repositories that we have forked from the main ROS 2 repositories (e.g., rcutils and rcl). More on these repositories and on the changes we implemented can be found in the sections below.

## 5.1 Modifications to ROS 2

The aim of the modifications is to improve in overall ROS 2 performance. These modifications come from different sources, for instance, the ROS 2 Working Groups or parties that are interested in improving a specific ROS 2 aspect. In our case, the consortium rose technical discussion in fields that are related to microcontrollers, such as real-time determinism, memory footprint or memory allocation and fragmentation.

As ROS 2 has a layered stack, it is important to understand how each layer behaves and affects the overall performance. The use of test tools is the right method for gaining this comprehension. Thanks to the conclusions we can take from these tests, ROS 2 modifications and improvements can be proposed to the rest of ROS 2 developers.

The work package five gives the tools to the members for analyzing micro-ROS. In addition, consortium members also allocated some time to investigate tools for ROS 2 analysis. ROS 2 developers can use these tools for their daily tasks and, once the project is over, it is expected that the community should have adopted them, as it will be of their own interest to maintain and extend them.

| Tool/asset/package | Software Layer | OFERA Maintainer | Long-term maintainer Candidate |
|---|---|---|---|
| Benchmarking tools | Other | PIAP | Open Robotics, Community |
| ROS 2 tracing instrumentation | Other | Bosch | Open Robotics, Community |
| ROS 2 trace analysis | Other | Bosch | Bosch |

Another valuable assessment that has been undertaken by the consortium (eProsima, specifically) is the assessment of the memory footprint of both the micro-ROS stack and of its middleware implementation, the Micro XRCE-DDS library. The relevant results can be found here and here.

- **Bencharking tools**

The platform that PIAP provides allows to test the complete micro-ROS stack, starting from the RTOS itself, up to the application user wants to deploy in its robot. During the project period, this task is of PIAP's competence. But, as these tools are useful for the entire ROS 2 community, it is expected that the OSRF and/or the Embedded Working Group take over and further extend and maintain them. The benchmarking tools are intended for embedded software benchmarking with low/none overhead intrusion. These tools are relatively easy to use and allow simple analysis of the obtained data also by graphic visualizations.

Information for the community on these benchmarking tools and on the results obtained with them can be found here and here.

- **ROS 2 tracing instrumentation and analysis**

The aim of this tool is to log the ROS 2 execution, to analyze it afterwards. Similar to earlier work for ROS 1, the instrumentation itself belongs into the ROS 2 core packages, which are maintained by Open Robotics. The maintainers have accepted the corresponding pull requests – most important ros2/rcl#473 and ros2/rclcpp#789, followed by several smaller ones – end of 2019 and beginning of 2020.

Separate from that, a number of default analysis scripts are provided at gitlab.com/ros-tracing/tracetools_analysis. They are maintained by members of the ROS Real-time Working Group.

## 5.2 Added Features within the Core Libraries

In the design process of ROS 2, many features that are wanted have been added to the development roadmap. These features are being developed gradually, as the effort is big and priorities of those features varies. Part of the software packages that are being implemented in OFERA project, contributes to that list. In addition, its implementation takes in mind its use in micro-ROS, tackling both implementations.

The Work Package 4 of the OFERA project is the one providing this kind of software entities, such as System-Modes, Diagnostics, and Executors.

As these software packages will supplement the ROS 2 core, it is expected that the maintenance will be performed by Open Robotics. If there is desire of extending those capabilities, the interested party should contribute and develop it.

| Tool/asset/package | Software Layer | OFERA Maintainer | Long-term maintainer Candidate |
|---|---|---|---|
| Callback-group-level Executor + Demo | Client library | BOSCH | ROS Client Library WG |
| System modes | Client library | BOSCH | BOSCH |
| Diagnostics | Client library | BOSCH | BOSCH |
| Client library extensions | Client library | eProsima | eProsima |

- **Callback-group-level Executor**

The ROS 2 Executor concept is still under discussion, despite several releases and contributions from the community. In May 2019, a Real-Time Working Group (WG) has been founded on initiative of Open Robotics and the ROS 2 TSC, with the goal to develop robust, real-time execution mechanisms for ROS 2. Several OFERA partners participate in this WG to ensure that the Executor concepts developed in micro-ROS are aligned with the concepts and implementations for micro-ROS and with the ultimate goal to bring them in the mainline code of rclcpp and rclc in the ROS 2 organization on GitHub.

In September 2020, developers from Open Robotics merged the Callback-group-level Executor, which is a refinement of Executor interface developed in OFERA in 2018, into the mainline code of rclcpp (cf. ros2/rclcpp#1218). It is currently maintained by Open Robotics and in future by a new working group, the ROS Client Library WG.

To demonstrate the use of the Callback-group-level Executor, OFERA members from Bosch open-sourced the *examples_rclcpp_cbg_executor* package in the ros2/examples repository. In future, this will be also maintained by the ROS Client Library WG.

- **System modes**

The system modes package is developed in OFERA based on the new ROS 2 lifecycle, which is part of rclcpp and maintained by Open Robotics. Since Bosch plans to use and develop this repository further after the end of OFERA, it will be also maintained by Bosch.

- **Diagnostics**

The diagnostics package of ROS 1 is widely used. Although it belongs to the core ROS organization on GitHub, it is not maintained by Open Robotics but an external developer. In the OFERA project, Bosch ported diagnostics to ROS 2 in 2019 and 2020 as an important dependency for further works on the system modes concept and for micro-ROS-specific diagnostics mechanisms. The ported repository has been stored in the ROS organization by Open Robotics on GitHub, at github.com/ros/diagnostics. Bosch plans to maintain this repository in the long-term.

- **Client library extensions**

At its beginnings, the OFERA consortium has agreed to stick with the current existing ROS 2 client libraries, rcl (with modules), rcutils, rclcpp, and rosidl_typesupport, with the addition of custom parts of code needed for exposing functionalities required by the nature of the middleware used in the lower layers, mostly configurations. The OFERA partner eProsima has been and is actually in charge of extending the existing rcl library to expose the required APIs.

On the other side, the initial decision of reusing the rclcpp modules has been progressively discarded given that this language is largely unsupported into resource-constrained devices in favour of the implementation of higher-level concepts such as executors and lifecycle into rclc, see below.

Currently, we work on bringing all changes to rcl, rcutils, and rosidl_typesupport back into the main repositories in the ROS 2 organization on GitHub.

## 5.3 Specific Software

As the aim of the OFERA project is to support ROS 2 in microcontrollers, there is code that only belongs to these resource-constrained devices. If we compare the ROS 2 and micro-ROS stacks, we can see that the bottom part is where the most substantial differences are located. For example, micro-ROS makes use of a RTOS, which is not contemplated in ROS 2.

To minimize the support effort, it is of micro-ROS interest to keep those software entities that live outside ROS 2 as few as possible. The following table shows which are those software entities and which is the expected long-term maintainer:

| Tool/asset/package | Software Layer | OFERA Maintainer | Long-term maintainer Candidate |
|---|---|---|---|
| FreeRTOS | RTOS | eProsima | FreeRTOS Community & EWG |
| Zephyr RTOS | RTOS | eProsima | Zepyr Community & EWG |
| NuttX RTOS | RTOS | No current maintainer | NuttX Community & EWG |
| Micro XRCE-DDS | Middleware | eProsima | eProsima |
| `rmw` for Micro XRCE-DDS | Middleware | eProsima | eProsima |
| micro-ROS to FIWARE bridge | Other | eProsima | eProsima |
| Micro XRCE-DDS TypeSupport | Client Library | eProsima | eProsima |
| rclc | Client Library | BOSCH, eProsima | BOSCH, eProsima |
| micro-ROS diagnostics | Client Library | BOSCH | BOSCH |

- **FreeRTOS**

micro-ROS has started supporting FreeRTOS around the end of 2019, but it was included officially among the supported RTOSes only in the Foxy release of the summer of 2020. As this RTOS is very popular and of widespread interest for the embedded community, eProsima in in charge of giving support to it while the project lasts (that is, until December 2021). It is then expected

that as the project gains more and more users' backup and momentum from the community, such support will be undertaken by this very community accordingly.

- **Zephyr RTOS**

micro-ROS has started supporting Zephyr at the beginning of 2020, but it was included officially among the supported RTOSes only in the Foxy release of the summer of 2020. As this RTOS is very popular and of widespread interest for the embedded community, eProsima in in charge of giving support to it while the project lasts (that is, until December 2021). It is then expected that as the project gains more and more users' backup and momentum from the community, such support will be undertaken by this very community accordingly.

- **NuttX RTOS**

NuttX, even though selected as the project official RTOS in the beginning, is not being supported by any specific partner after Acutronic Robotics left the project. It actually consists of a maintained fork of the original BitBucket NuttX repository. The fork ensures the support of the development platforms selected in the project. Nowadays, there is a great divergence between these repositories.

- **Micro XRCE-DDS**

eProsima Micro XRCE-DDS is an implementation of the OMG DDS-XRCE standard. This middleware implementation enables the microcontrollers to use DDS, thus enabling micro-ROS to interoperate with ROS2 natively.

The nature of DDS does not make it usable in resource-constrained environments like microcontrollers, due to computation and memory restrictions. Addressing this issue, the OMG designed a new communication standard called DDS-XRCE. This standard allows small devices to communicate with DDS. The central architectural concept is the existence of tiny clients communicating with a relatively more prominent Agent, which have DDS capabilities. In such architecture, the Agent acts in DDS world on behalf of the clients.

eProsima is the developer of a first and open source implementation, called Micro XRCE-DDS. This implementation, apart of being used as part of micro-ROS layers, it is used in other different projects not related to micro-ROS, making it a valuable product for eProsima.

eProsima treats Micro XRCE-DDS as any other of their products providing customer support and continuously maintaining and improving it, both with the community and with internal developments.

Currently, all the Micro XRCE-DDS packages are hosted in the eProsima GitHub and a Pull Request has already been made to the REP-2004 of ROS 2 so as to include these packages within the common ROS 2 packages.

- **`rmw` for Micro XRCE-DDS**

Mimicking ROS 2 architecture, an abstraction layer over the middleware is present in micro-ROS. This layer, RMW, is in charge of abstracting upper layers from the middleware implementation underneath even from the protocol used, DDS in ROS 2, DDS-XRCE in micro-ROS. eProsima developed a specific implementation of this layer for its DDS-XRCE implementation, Micro XRCE-DDS. eProsima developed this layer the same way as the Fast DDS implementation layer is in ROS 2. eProsima will integrate this as a regular ROS 2 package and keep it up to date related to:

1) Future Micro XRCE-DDS revisions due to implementation or DDS-XRCE standard changes.
2) RMW abstraction revisions from Open Robotics changes.

Currently, the rmw_microxrcedds package is hosted in a micro-ROS repository but, like the rest of micro-ROS packages it has been proposed to be integrated within ROS 2 repositories.

- **micro-ROS to FIWARE bridge**

eProsima is the owner of SOSS, a component to bridge several incompatible protocols, including the FIWARE Context Broker and Fast DDS, and ROS 2. As part of the OFERA project, eProsima has implemented the software components needed to intercommunicate FIWARE with ROS 2, and FIWARE with micro-ROS. This tool is developed and maintained by eProsima.

- **Micro XRCE-DDS TypeSupport**

Mimicking ROS 2 architecture, an typesupport for the middleware is present in micro-ROS. This layer is in charge of handling types serialization and deserialization in the middleware implementation, DDS-XRCE in micro-ROS. eProsima developed a specific implementation of this layer for its DDS-XRCE implementation, Micro XRCE-DDS. eProsima will integrate this as a regular ROS 2 package and keep it up to date related to:

1) Future Micro XRCE-DDS revisions due to implementation or DDS-XRCE standard changes.
2) RMW abstraction revisions from Open Robotics changes.

Currently, the rosidl_typesupport_microxrcedds package is hosted in a micro-ROS repository but, like the rest of micro-ROS packages it has been proposed to be integrated within ROS 2 repositories.

- **rclc**

Bosch and eProsima are in charge of implementing a feature-complete Client Library in the C programming language tailored to the needs of resource-constrained devices. Currently, the relevant repository can be found at github.com/micro-ROS/rclc. Most important, it provides the rclc Executor, but also mechanisms for parameters, graph, runtime lifecycle, etc.

After negotiations with Open Robotics, in March 2020, Bosch and eProsima together took over the maintainership of the rclc repository in the ROS 2 organization at github.com/ros2/rclc and can thus upstream all improvements and features from micro-ROS/rclc themselves. We also plan to have rclc declared a core package under REP-2005, which ensures that it is considered by developers and maintainers of the lower layers in case of API breaking changes.

- **micro-ROS diagnostics**

The micro-ROS-specific diagnostics functionality (which comprises very few source files only) is stored at the micro-ROS organization at github.com/micro-ROS/micro_ros_diagnostics and maintained by Bosch.

# 6 Conclusion

To sum up, we can see that the feasibility of introducing the developed software in OFERA project into ROS 2 is different, depending on the nature of each software entity. Taking this into account, we have set goals to try to merge them into ROS 2 development and maintenance mechanisms, following different strategy for each group.