D3.15

FIROS2
Software new release Y3

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D3.15
Deliverable name	FIROS2
Date	June 2020
Dissemination level	public
Workpackage and task	WP3 - Task 3.4
Author	Borja Outerelo Gamarra (eProsima)
Contributors	José Antonio Moral Parras (eProsima)
Keywords	micro-ROS, ROS2, FIWARE, intercommunication, NGSIv2 protocol, communication bridges, context broker, SOSS, System Handler
Abstract	This document provides links to the released software and documentation for deliverable D3.15 *FIROS2_new_Release* of the Task 3.4 *FIWARE Interoperability*.

Contents

1 Acronyms and keywords

Term	Definition
IS	Integration Service
FIROS2	FIWARE - ROS2 bridge
FTP	Focused Technical Project
OSRF	Open Source Robotics Foundation
ROS2	Robot Operating System 2
SOSS	System of Systems Synthesizer
XTypes-DDS	Extensible and Dynamic Topic Types for DDS

2 Summary

As explained in *D3.12 Interoperability Report - Y3* and in *D1.2 Annual Project Report - Y2*, when it comes to the previous and the current year, the efforts made with respect to task 3.4 have been focused on aligning FIROS2 with the development of the SOSS tool. SOSS is a systems integration platform developed as a joint effort between eProsima and OSRF and with the help of a FTP from the EU-funded project ROS-Industrial.

In this document, we introduce basic concepts and usage of the FIROS2 tool, which is nothing but the specialization of SOSS tool to communicate two specific middlewares: micro-ROS and FIWARE Context Broker. This is just one of the many use cases that SOSS is able to solve seamlessly, thanks to its ability to put into communication any middleware whose System Handle has been implemented with the rest of the supported middlewares by the SOSS integration tool. An up-to-date list of the supported middlewares by SOSS platform can be found here.

3 FIROS2 bridge

3.1 Justification

FIROS2 bridge comes from the intention of communicating micro-ROS with FIWARE Context Broker, so that interoperability between applications relying on both middlewares can be achieved seamlessly. To accomplish this, two different approaches can be taken:

- Implementing an ad-hoc bridging communication tool for translating FIWARE's messages into microROS (that is, ROS2) messages types, and viceversa.
- Relying on an integration platform that uses a common language representation that defines a conversion from/to the specific types of the middleware.

While the first approach might result in a more lightweight tool, it has several flaws, for instance a more difficult maintenance and the incapability of communicating with any other middleware, rather than ROS2 or micro-ROS. On the other hand, using an IS platform such as SOSS, enables automatically the possibility of communicating with a wide (and growing) set of middlewares, if their System Handle implementation is available.

3.2 SOSS

SOSS addresses the task of providing a common interface for communicating software platforms that speak different languages. It allows the user to use one of the supported plugins or *System Handles* for a specific middleware, such as DDS, ROS2, FIWARE or ROS. SOSS can act as an intermediate message-passing tool that, by speaking a common language, centralizes and mediates the integration. A SOSS instance is configured and launched by means of a YAML file, which allows the user to provide a mapping between the different topics and services that two or more applications, developed using different middleware platforms, can exchange information about. Users can also develop their own System Handles for a new middleware, automatically enabling communication capabilities with all the rest of supported middlewares.

Usually, types are defined using a common language representation, used by SOSS to create a shared representation of the exchanged information, so that it can be processed, converted and remapped to every middleware's types implementation, when required. This common representation is provided, user-wise, using IDL definitions, which are parsed and converted into Dynamic Types representations at runtime, using eProsima's XTypes-DDS implementation.

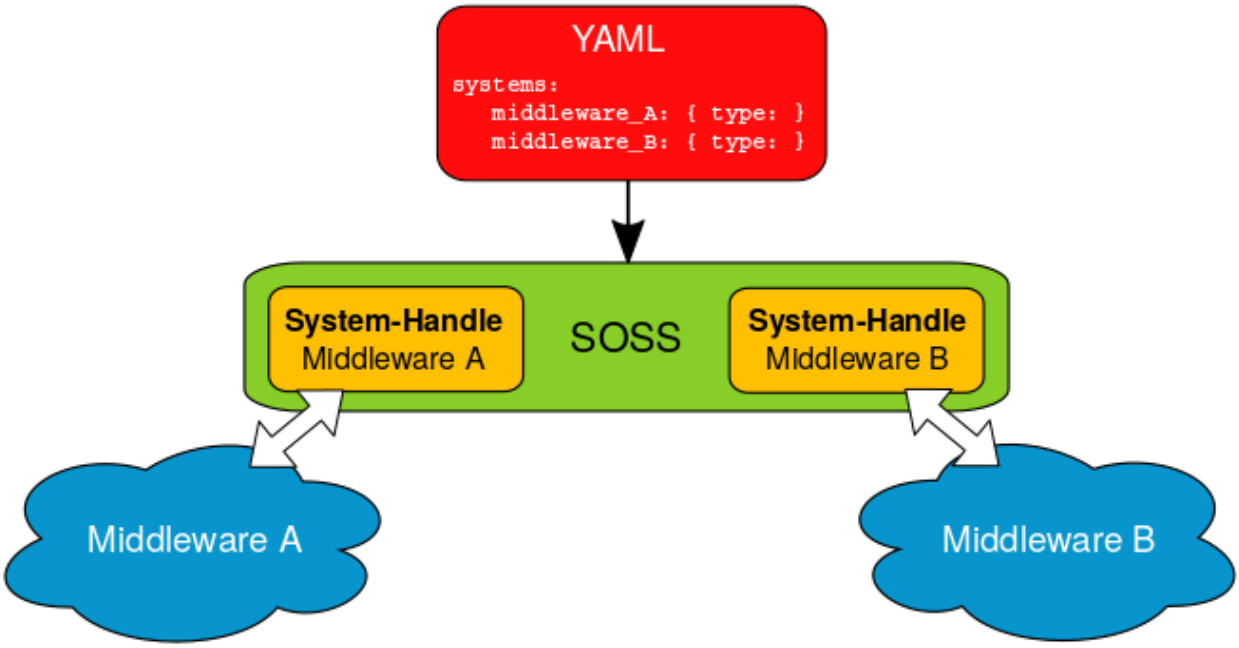

Figure 1: SOSS architecture

3.3 FIWARE System Handler

The FIWARE System Handle allows bringing information from and to FIWARE's Context Broker into the SOSS world. This System Handle is configured and launched the same way as any SOSS System Handle. For a complete description of the available parameters and entities which can be configured in a certain SOSS instance, refer to this section of the SOSS documentation. Besides the

standard information included in any System Handle's configuration (such as system's name and type, which would be `fiware` for this specific System Handle), in the case of the FIWARE System Handle users must specify two extra YAML key-value pairs, which are the host's IP and port in which this System Handle will try to connect to an instance of FIWARE's Orion Context Broker.

Regarding more specific details about the implementation, FIWARE does not allow certain characters in its entities names. For this reason, if a type defined in the topics section of the configuration file has in its name a `/`, the FIWARE System Handle will map that character into two underscores. This is something important to notice when connecting to ROS2, because in ROS2 most of the types have a `/` in their names. To deal with this issue, using SOSS remapping capabilities come in handy.

4 Overview to Results

This document provides links to the released software and documentation for deliverable D3.13 *FIROS2_new_Release* of the Task 3.4 *FIWARE Interoperability*.

5 Links to Software Repositories

Users can find the latest versions of SOSS-FIWARE System Handle in its GitHub repository:

- Git repository: SOSS-FIWARE
  Branch: feature/xtypes-support
  Commits: 01d8f3f

User will also need SOSS and the ROS 2 System Handler, both hosted in:

- Git repository: SOSS
  Branch: feature/xtypes-dds
  commits: 23e7d13
  ROS 2 System handler: /packages/ros2

6 Annex 1: Hands-on with FIWARE System Handle: compilation and use case

6.1 Installation

To install this package into a workspace already containing SOSS, just clone this repository into the sources directory and build it:

```
1 $ cd <soss workspace folder>
2 $ git clone https://github.com/eProsima/SOSS-FIWARE.git src/soss-fiware
3 $ colcon build --packages-up-to soss-fiware
```

If you do not know how to create a SOSS workspace from scratch, refer to this section of the SOSS documentation.

6.2 Use case - Connecting with ROS2

0. Prerequisites: curlpp and asio installed

1. Create a colcon workspace.

```
$ mkdir -p soss_wp/src
$ cd soss_wp
```

2. Clone the soss project into the source subfolder.

```
$ git clone https://github.com/osrf/soss_v2.git src/soss --branch feature/xtypes-dds
```

3. Clone this project into the subfolder.

```
$ git clone https://github.com/eProsima/SOSS-FIWARE.git src/soss-fiware --branch feature/xtypes-dds
```

   The workspace layout should look like this:

```
    soss_wp
    ...src
    ......soss
    .........(other soss project subfolders)
    .........packages
    ............soss-ros2 (ROS2 system handle)
    ......soss-fiware (repo)
    ............fiware (soss-fiware colcon pkg)
    ............fiware-test (soss-fiware-test colcon pkg)
```

4. In the workspace folder, execute colcon:

```
$ colcon build --packages-up-to soss-fiware
```

5. Source the current environment:

```
$ source install/local_setup.bash
```

6.3 Usage

This System Handle can be used to connect FIWARE with other systems.

6.3.1 Configuration

SOSS must be configured with a YAML file, which tells the program everything it needs to know in order to establish the connection between two or more systems that the user wants. For example, if the user wants to exchange a simple string message between FIWARE and ROS2, the configuration file for SOSS should look as follows.

```
systems:
    ros2: { type: ros2 }
    fiware: { type: fiware, host: CONTEXT_BROKER_IP, port: 1026}

routes:
    fiware_to_ros2: { from: fiware, to: ros2 }
    ros2_to_fiware: { from: ros2, to: fiware }

topics:
    hello_fiware: { type: "std_msgs/String", route: ros2_to_fiware }
    hello_ros2: { type: "std_msgs/String", route: fiware_to_ros2 }
```

6.3.2 More information

- For more information, you can see the demo steps and the related video
- Also, you can have a look at the internal design
- For a fast configuration, you can use the dockerfile. **NOTICE**: Fiware Orion Context Broker may not be able to reach the docker IP, as it is not accessible outside the host computer by default. You can run (or build if necessary) the docker with –network=host to share the network interface of the host with Docker, and make it accessible in your LAN.

6.3.3 Changelog

6.3.3.1 v0.2.0

- Added dockerfiles
- Added integration tests
- Removed asio as local thirdparty
- Subscription host and port automatically generated if they are not specified.

6.3.3.2 v0.1.1

- Fiware communication take into account the topic type
- Added thread protection

6.3.3.3 v0.1.0

- Fiware communication in both directions based on topic

Supported by ROSIN - ROS-Industrial Quality-Assured Robot Software Components. More information: * http://rosin-project.eu