

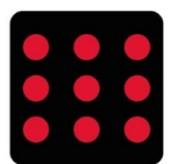
ROS2

for
ROS Developers

ROS



2



Intelligent
Robotics
Lab

Prof. Dr. Francisco Martín Rico
Madrid 30-1-2020



Universidad
Rey Juan Carlos

Introduction



ROS (Robot Operating System)

“The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.”

<http://www.ros.org/>



History

- ROS was born in 2006 at Stanford, within STAIR project
- Willow Garage in 2007
- Open Source Robotics Foundation in 2013
- Since 2013 is considered the standard in Robotics



ROS Distributions

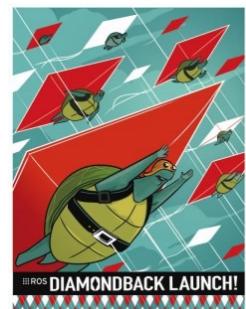


Mar 2, 2010

ROS Box Turtle



Aug 2, 2010



Mar 2, 2011



Aug 30, 2011



Apr 23, 2012



Dec 31, 2012



Sept 4th, 2013



Jul 22nd, 2014



May 23rd, 2015



May 23rd, 2016



May 23rd, 2017



May 23rd, 2018

Resources



- [ROS.org](#) [link]
- [Distributions](#) [link]
- [Package documentation](#) [link] [link]
- [ROS Wiki](#) [link]
- [ROS Answers](#) [link]
- [ROS Discourse](#) [link]
- [Blog](#) [link]



[What is ROS?](#)
The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

[Read More](#)

[ROS Melodic Morenia](#)
Melodic Morenia is the 12th official ROS release. It is supported on Ubuntu Artful and Bionic, along with Debian Stretch. Get Melodic Morenia now!

[Download](#)

[ROS Kinetic Kame](#)
Kinetic Kame is the 10th official ROS release. It is supported on Ubuntu Wily and Xenial. Get Kinetic Kame now!

[Download](#)

[Wiki](#)
Find tutorials and learn more

[ROS Answers](#)
Ask questions. Get answers

[Blog](#)
Get the latest news

ROS Resources: Documentation | Support | Discussion Forum | Service Status | Q&A answers.ros.org

ROS Discourse

Categories Topics Latest New (1) Unread (5) Top + New Topic

Category	Topics	Lastest
General	896	1 unread
ROS	896	0
Autoware	171	1 unread
Next Generation ROS	469	3 unread
Quality Assurance	62	1 new

[tags](#) [users](#) [badges](#)

Hi there! Please sign in | help

TF Lookup would require extrapolation into the future

4 t robot_localization gps

I would appreciate some guidance to understand what may be going on here.
I am trying to use the very nice robot_localization package to integrate GPS, IMU, Optical Flow and a few other sensors to localize a drone.
When running my configuration with sensor data coming from a recorded bag file, I get the following error from the tf_localization_node node:

[WARN] [1406560584.464395417, 1406223582.586891248]: Could not obtain transform from utm to nav. Error was Lookup would require extrapolation into the future. Requested time 1406223582.213000000 but the latest data is at time 1406223582.576022606, when looking up transform from frame [utm] to frame [nav]

I've been trying to understand why this is happening and I've gathered the following information:

- The CPU usage is pretty low. I don't think this could be happening because of lack of processing capacity
- What the code seems to be trying to do is using lookupTransform to calculate TF using the timestamp of the GPS message received
- The transformation between [utm] and [nav] is being published at 50Hz. GPS messages are being published at 1Hz.
- I can see GPS messages being published with timestamps matching the warning messages. I can also see TF publishing transformations matching the timestamp of "the latest data is at time ..." but also many more later messages:

```
stamp:
secs: 1406223582
nsecs: 576022606
frame_id: nav
child_frame_id: utm
transform:
translation:
```

ROS.org

About | Support | Discussion Forum | Service Status | Q&A answers.ros.org

Search: Submit

Documentation Browse Software News Download

navigation

Indigo | Kinetic | Lunar | melodic Show EOL distros:

Documentation Status

navigation: amcl | base_local_planner | carot_planner | clear_costmap_recovery | costmap_2d | dwa_local_planner | fake_localization | global_planner | map_server | move_base | move_base_msgs | move_slow_and_clear | nav_core | navfn | rotate_recovery | voxel_grid

Package Summary

✓ Released ✓ Continuous Integration: 81 / 81 ✓ Documented

A 2D navigation stack that takes in information from odometry, sensor streams, and a goal pose and outputs safe velocity commands that are sent to a mobile base.

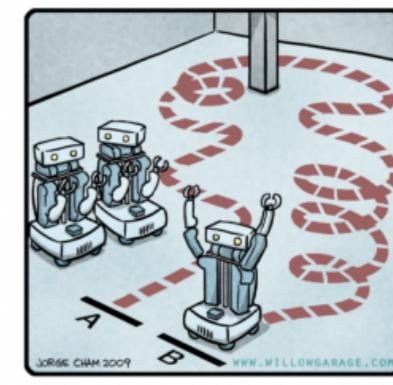
- Maintainer status: maintained
- Maintainer: Michael Ferguson <mfergus7 AT gmail DOT com>, David V. Lutz <davidvlutz AT gmail DOT com>, Aaron Hoy <aoy DOT fetcrobotics DOT com>
- Author: contradic@Gmail.com, Eltan Marder-Eppstein
- License: BSD, LGPL, GPL (amcl)
- Source: git <https://github.com/ros-planning/navigation.git> (branch: melodic-devel)

Table of Contents

1. Overview
2. Hardware Requirements
3. Documentation
4. Report a Bug
5. Examples
6. Tutorials
 1. Basic ROS Navigation Tutorials
 2. Navigation Tutorials for the Care-O-bot
 3. Navigation Tutorials for the TurtleBot
 4. Navigation Tutorials for Husky
 5. Navigation Tutorials for the MRPP2
 6. Navigation Tutorials for evanbot
 7. MRPT Navigation Tutorials
 8. Related Publications

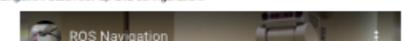
Available Translations: SimpleChinese

R.O.B.O.T. Comics



1. Overview

The Navigation Stack is fairly simple on a conceptual level. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. Use of the Navigation Stack on an arbitrary robot, however, is a bit more complicated. As a pre-requisite for navigation stack use, the robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level. To help with this process, this manual is meant to serve as a guide to typical Navigation Stack set-up and configuration.



Limitations

- Centralized in Master
- Own networking implementation
 - No real multicast
 - No QoS
- No multirobot
- No Real Time
- No multiplatform
- Important components are patches
- Only C++ and Python independent implementations
- Security



ROS2 Distributions



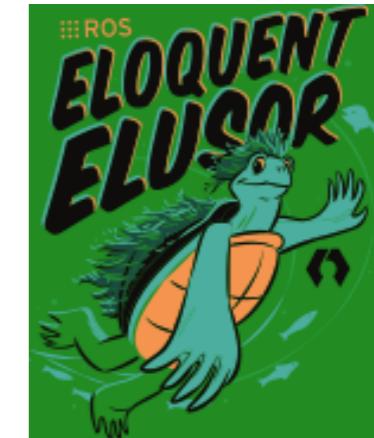
ROS2 Distributions



Dec 8th, 2017



Dec 14th, 2018



Nov 22nd, 2019



July 2nd, 2018



May 31st, 2019

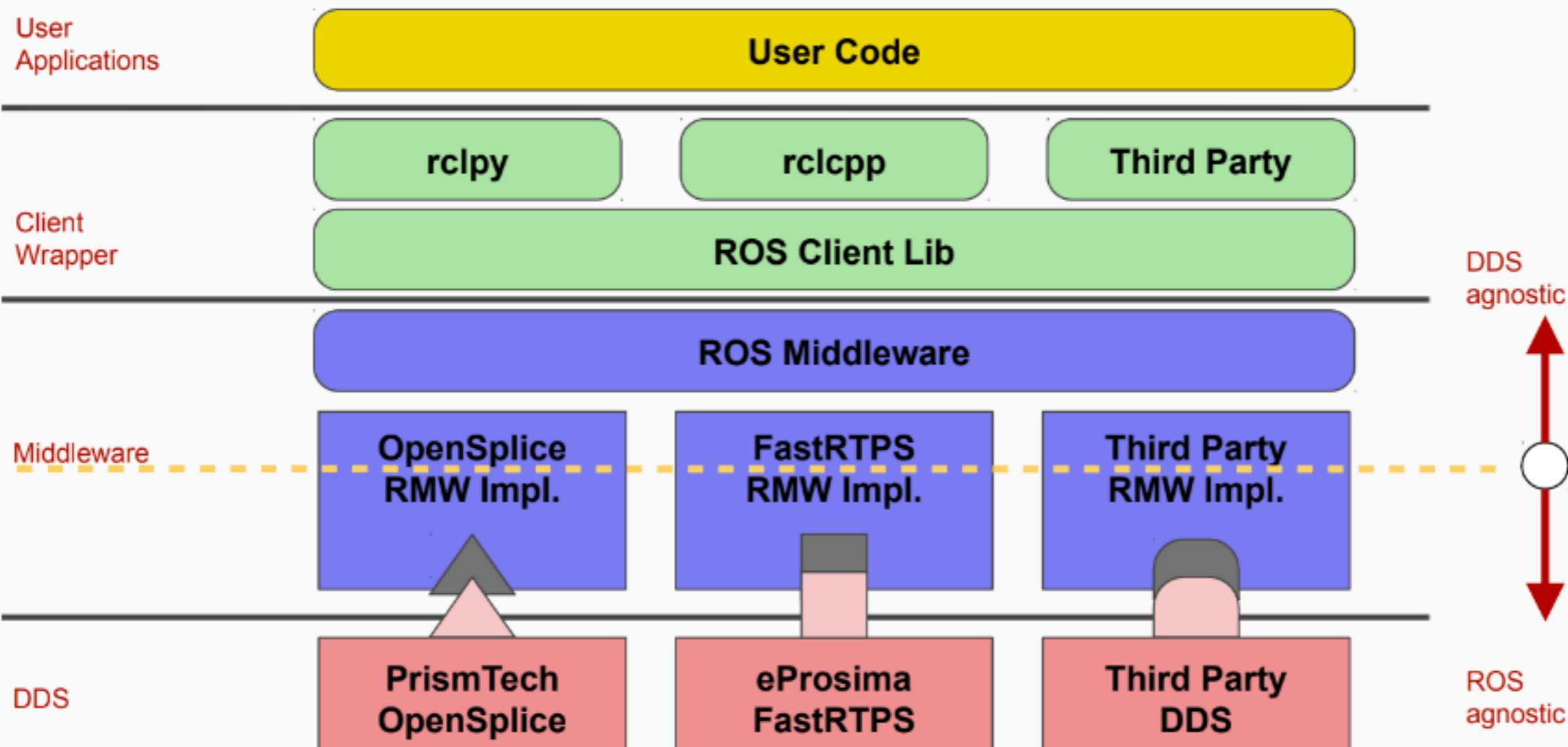
Main Design

ROS2

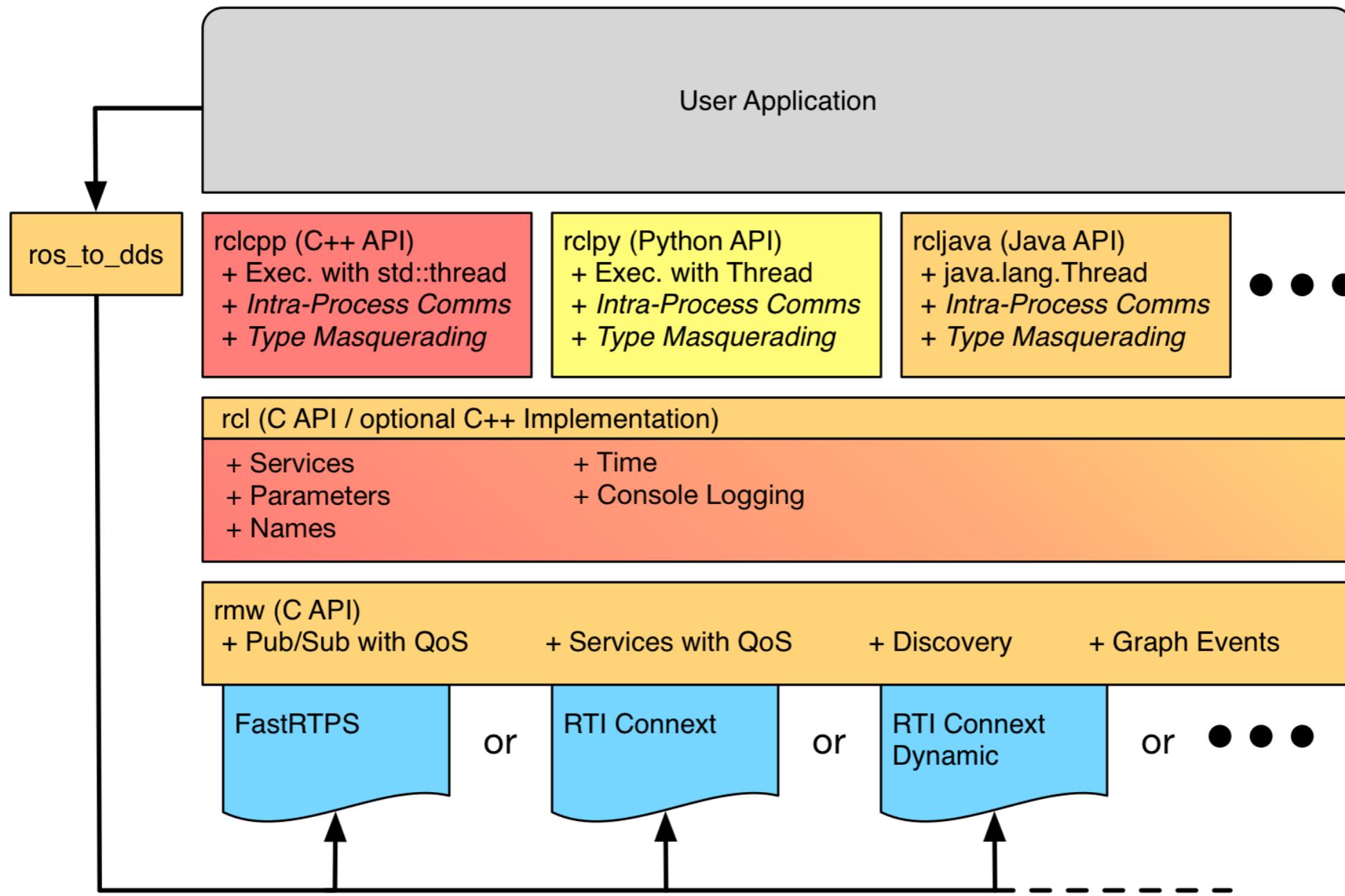


Design

ROS 2.0 Architecture Overview



Design



* *Intra-Process Comms* and *Type Masquerading* could be implemented in the client library, but may not currently exist.

Hands on!!

ROS2



ROS2 Setup



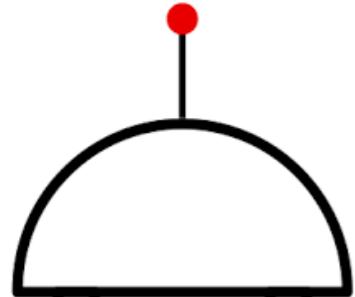
- Install ROS2

<https://index.ros.org/doc/ros2/Installation/>

- Setup ROS2

```
source /opt/ros/eloquent/setup.bash  
source /home/fmrico/ros/ros2/ros2talk_ws/install/setup.bash  
export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
```

Virtual machine with ROS2



GRUPO DE ROBÓTICA

robotica.unileon.es

A screenshot of a Linux desktop environment. In the center, there is a terminal window titled "Terminal - root@daaa850e47c3: ~/ros_ws#". It shows a command-line prompt. To the left of the terminal is a code editor window titled "Project - ~/ros_ws - Atom". The code editor displays a C++ file named "composable_node_pub_timer.cpp". The file contains code for a ROS 2 node that publishes messages at a specific rate and logs them. The code editor interface includes a sidebar for navigating through files and a status bar showing the file path and line numbers.

```
// See the License for the
// limitations under the License.
#include <chrono>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.h"
using namespace std::chrono_literals;

class MyNodePublisher : public rclcpp::Node {
public:
    MyNodePublisher(const std::string &name) : Node(name), counter_(0) {
        pub_ = create_publisher<std_msgs::msg::String>(
            "hello_world", 10);
        timer_ = create_wall_timer(
            100ms, std::bind(&MyNodePublisher::timer_callback, this));
    }
    void timer_callback() {
        std_msgs::msg::String message;
        message.data = "Hello, world! " + std::to_string(counter_++) + " from " + get_name();
        RCLCPP_INFO(get_logger(), "Publishing [%s]", message.data.c_str());
        pub_->publish(message);
    }
private:
    rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
    rclcpp::TimerBase::SharedPtr timer_;
    int counter_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<MyNodePublisher>("hello_publisher"));
    rclcpp::shutdown();
}
```

<https://roboticalabs.unileon.es:N/>

- Your ROS2 workspace
- colcon as build tool
- build, install, log, devel

rclcpp API

[intro_ros2/src/simple_node.cpp](#)

- Extensive use of advanced C++ (C++17 mostly)
- rclcpp::
- Nodes are no longer processes; they are objects
- No need for NodeHandle
- ros2 as super command

```
$ colcon build --symlink-install  
$ ros2 run intro_ros2 nodo_simple  
$ ros2 node list  
$ ros2 node info /simple_node
```

intro_ros2/src/simple_node_pub.cpp

- Messages includes change: lowercase and _
- Differentiation between msg, srv and action

```
$ ros2 msg list  
$ ros2 msg std_msgs/msg/String
```

← Deprecated

```
$ ros2 interface list -m  
$ ros2 interface show std_msgs/msg/String
```

- publishers and subscriptions are requested to nodes
- ROS_INFO -> RCLCPP_INFO
- Logger associated to a node id
- spin_some and spin_once (and more spinners)

```
$ ros2 run intro_ros2 simple_node_pub  
$ ros2 topic echo /chatter  
$ ros2 topic info /chatter  
$ ros2 topic hz /chatter  
$ ros2 topic bw /chatter
```

intro_ros2/src/simple_node_sub.cpp

[intro_ros2/src/simple_node_pub_qos.cpp](#)
[intro_ros2/src/simple_node_sub_qos.cpp](#)

```
auto subscription = node->create_subscription<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local(), callback);
```

```
auto publisher = node->create_publisher<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local());
```

[rclcpp QoS API](#)

QoS policies

History	Keep last	Keep all
Depth	Size	
Reliability	Best effort	Reliable
Durability	Transient local	Volatile

QoS profiles

Default	Reliable	Volatile	Keep last
Services	Reliable	Volatile	normal queue
Sensor	Best effort	small queue	
Parameters	Reliable	Volatile	Larger queue

QoS in ROS2



[intro_ros2/src/simple_node_pub_qos.cpp](#)
[intro_ros2/src/simple_node_sub_qos.cpp](#)

```
auto subscription = node->create_subscription<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local(), callback);
```

```
auto publisher = node->create_publisher<std_msgs::msg::String>(  
    "chatter", rclcpp::QoS(100).transient_local());
```

[rclcpp QoS API](#)

Compatibility of
QoS **durability**
profiles:

		Subscriber	
		Volatile	Transient Local
Publisher	Volatile	Volatile	No Connection
	Transient Local	Volatile	Transient Local

Compatibility of
QoS **reliability**
profiles:

		Subscriber	
		Best effort	Reliable
Publisher	Best effort	Best effort	No Connection
	Reliable	Best effort	Reliable

Better design for nodes



[intro_ros2/src/composable_node.cpp](#)

- Inherit from `rclcpp::Node`

[intro_ros2/src/composable_node_pub.cpp](#)

- Publishers/subscribers inside nodes

[intro_ros2/src/composable_node_sub.cpp](#)

- Multiples nodes per process

- **Executors**

- `rclcpp::executors::SingleThreadedExecutor` [API](#)
- `rclcpp::executors::MultiThreadedExecutor` [API](#)

More executions models



[intro_ros2/src/composable_node_pub_sub.cpp](#)

- `spin_until_future_complete`

[intro_ros2/src/composable_node_pub_timer.cpp](#)

- Timers for controlling rhythm
 - No extra threads - Graceful degradation

Parameters

- Parameters are stored in nodes
- They must be declared

parameters_ros2/src/simple_param_node.cpp

```
$ ros2 run parameters_ros2 simple_param_node  
$ ros2 param get /simple_param_node message
```

- You can subscribe to changes

parameters_ros2/src/reconf_param_node.cpp

```
$ ros2 run parameters_ros2 reconf_param_node  
$ ros2 param set /ReconfigurableNode speed 5.6
```

Launchers



- In Python
- More control

[intro_ros2/launch/composable_pub_timer_launch.py](#)

- `generate_launch_description` function that returns a `LaunchDescription` object

```
$ ros2 launch intro_ros2 composable_pub_timer_launch.py
```

- It contains actions: execute node, set env var, include other launcher and launcher args

[launchers_ros2/launch/first_launch.py](#)

[launchers_ros2/launch/second_launch.py](#)

- Set parameters from files

[parameters_ros2/launch/param_file.launch.py](#)

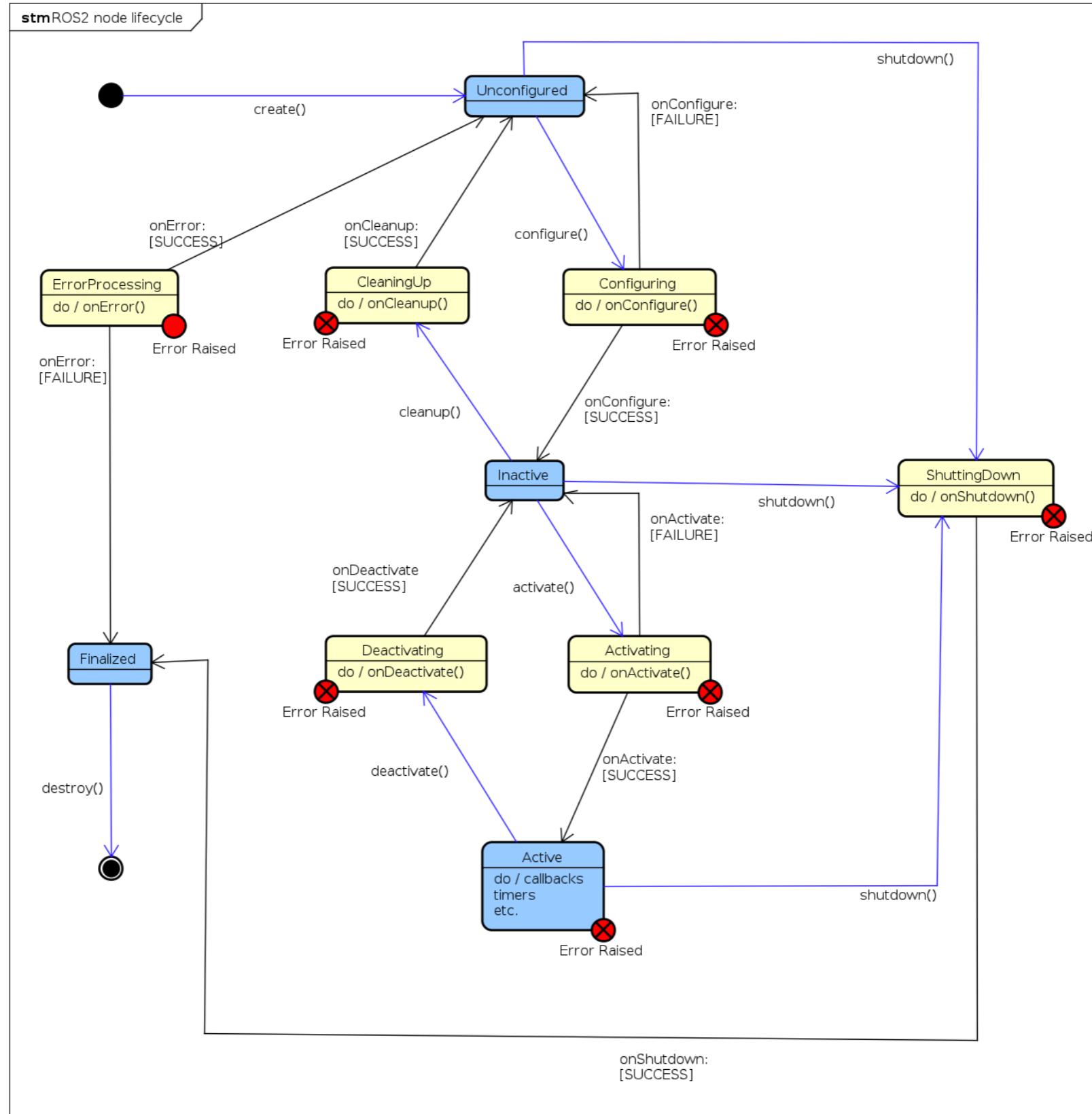
[parameters_ros2/config/config_2.yaml](#)

Lifecycle nodes



- In ROS, nodes are executing from initial
- Some nodes open devices
- Some nodes read params and configurations
- Specially critical for drivers
- In a launcher, there is not a way to control the boot order
- No way to pause, reconfigure, cleanup and shutdown nodes
- **ROS2: Lifecycle Nodes**
- LF nodes are in a state
- In each state, there are some available transitions
- Observed and commanded from outside (from launchers!!!)

States and Transitions



lifecycle_nodes_ros2/src/simple_lifecycle_node.cpp

- Inheritance from `rclcpp_lifecycle::LifecycleNode`
- Control of code in transitions
 - `on_configure`
 - `on_activate`
 - ...
- You can accept or reject a transition request

```
$ ros2 launch lifecycle_nodes_ros2 simple.lifecycle_node
$ ros2 lifecycle nodes
$ ros2 lifecycle get /lifecycle_node_example
$ ros2 lifecycle list /lifecycle_node_example
$ ros2 lifecycle set /lifecycle_node_example configure
$ ros2 lifecycle set /lifecycle_node_example activate
$ ros2 lifecycle set /lifecycle_node_example deactivate
```

- Lifecycle publishers

lifecycle_nodes_ros2/src/pub_lifecycle_node.cpp

Services and Actions



services_ros2/src/server_node.cpp

- Very similar to ROS

```
$ ros2 run services_ros2 server_node
$ ros2 service list
$ ros2 service type /reverse_string
$ ros2 interface show ros2talk_msgs/srv/ReverseString
$ ros2 service call /reverse_string ros2talk_msgs/srv/ReverseString
'{normal_sentence: "Hello World ROS2!!!!"}'
```

services_ros2/src/simple_client_node.cpp

- A little bit more complicated
- `spin_until_future_complete`

```
$ ros2 run services_ros2 server_node
$ ros2 run services_ros2 simple_client_node
```

actions_ros2/src/server_node.cpp

- Some callbacks
 - handle_goal
 - handle_cancel
 - handle_accepted
- Use of thread not mandatory
- Sending back feedback and result using goal_handle

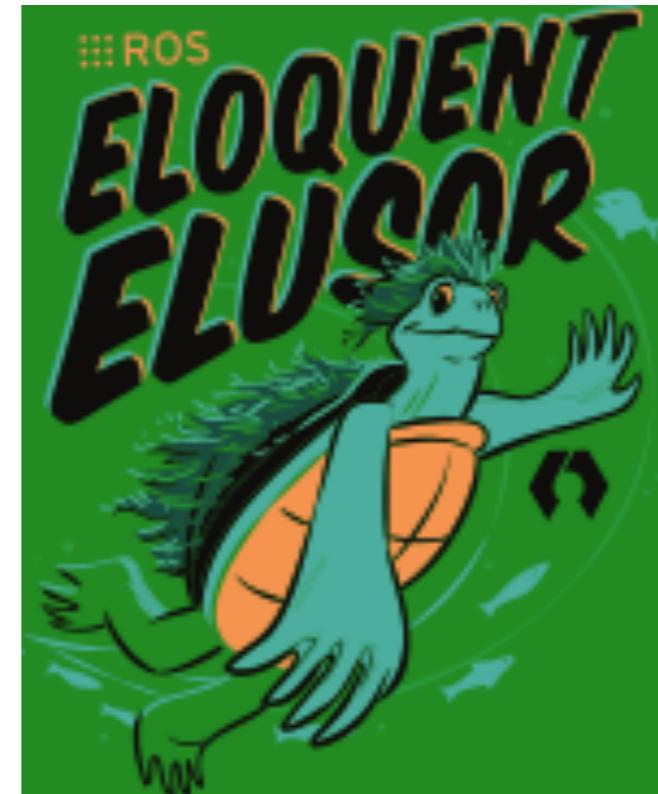
```
$ ros2 run actions_ros2 server_node  
$ ros2 action list  
$ ros2 action info /repeat_sentence
```

actions_ros2/src/client_node.cpp

- Callbacks feedback_callback result_callback

```
$ ros2 run actions_ros2 server_node  
$ ros2 run actions_ros2 client_node
```

Software in ROS2



Software growing up!!

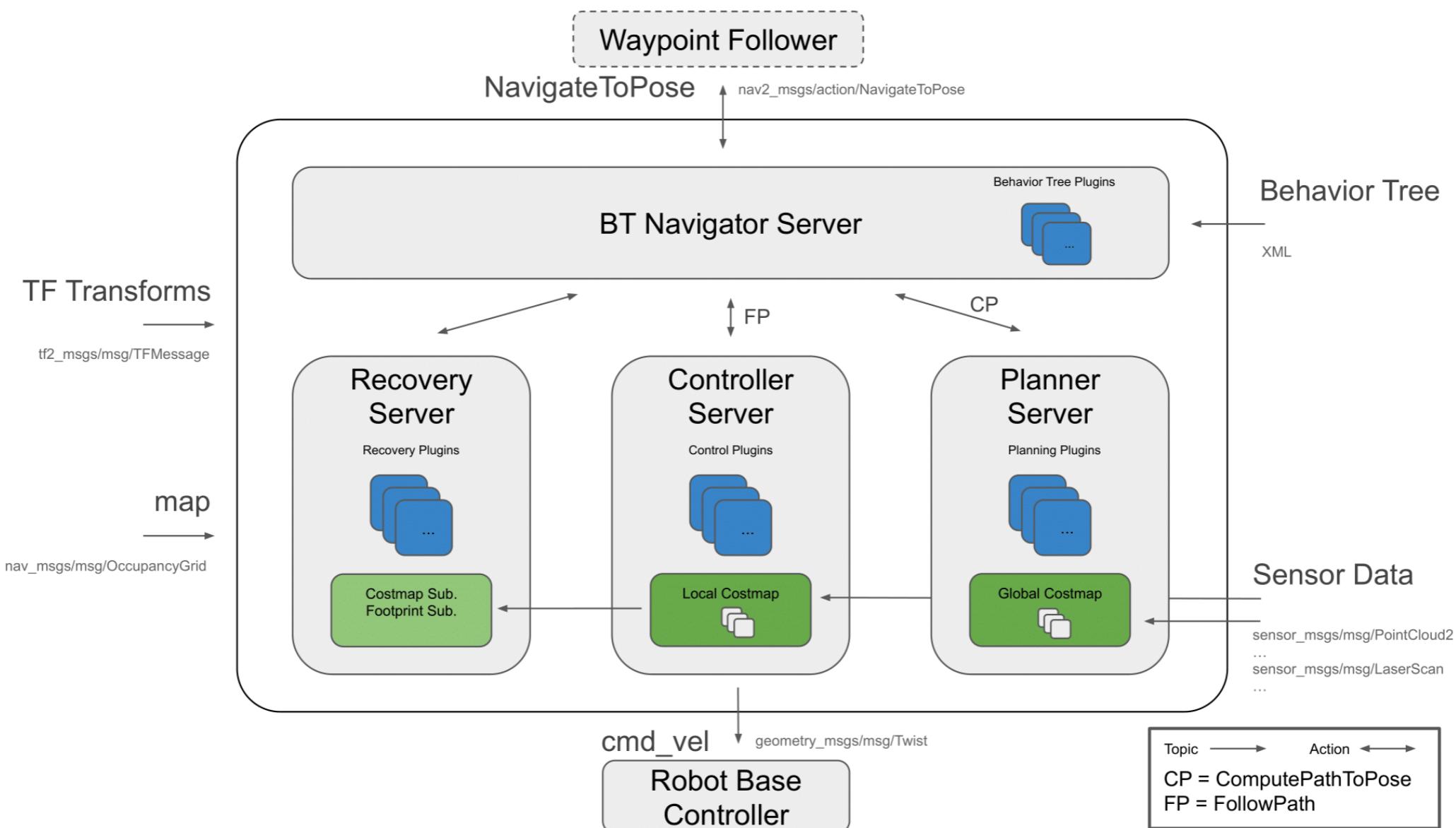
ROS → 2

- Growing last months
- Principal packages ported to ROS2
 - ImageTransport
 - TF2
 - Robots: Kobuki, TB3....
 - Drivers: Cameras, Lasers..
 - Behavior Trees
 - Navigation2
 - PlanSys2
 - ...

Navigation2



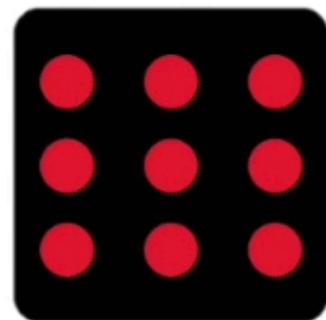
<https://github.com/ros-planning/navigation2>



Navigation2

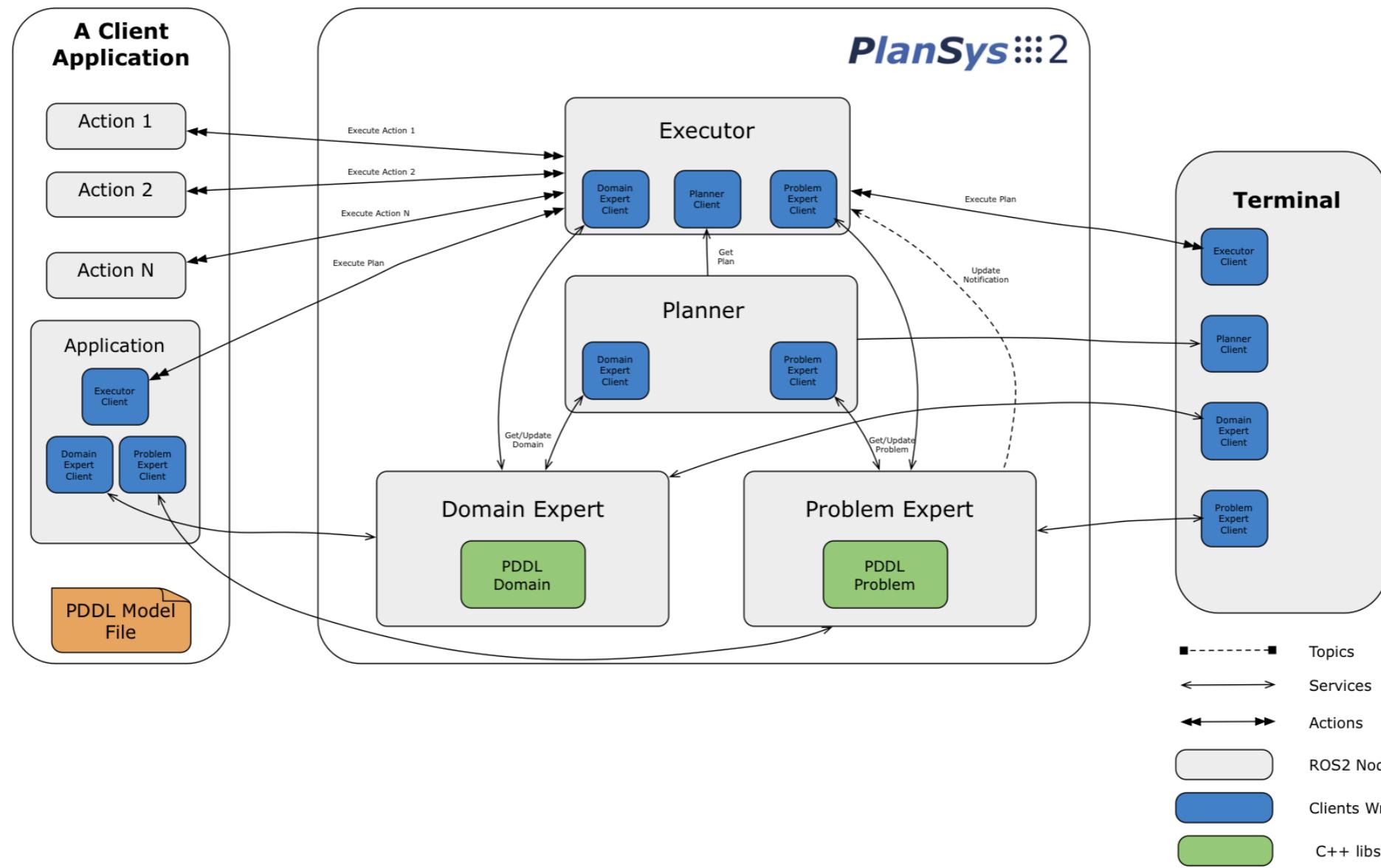
ROS → 2

<https://github.com/ros-planning/navigation2>



Intelligent Robotics *Lab*

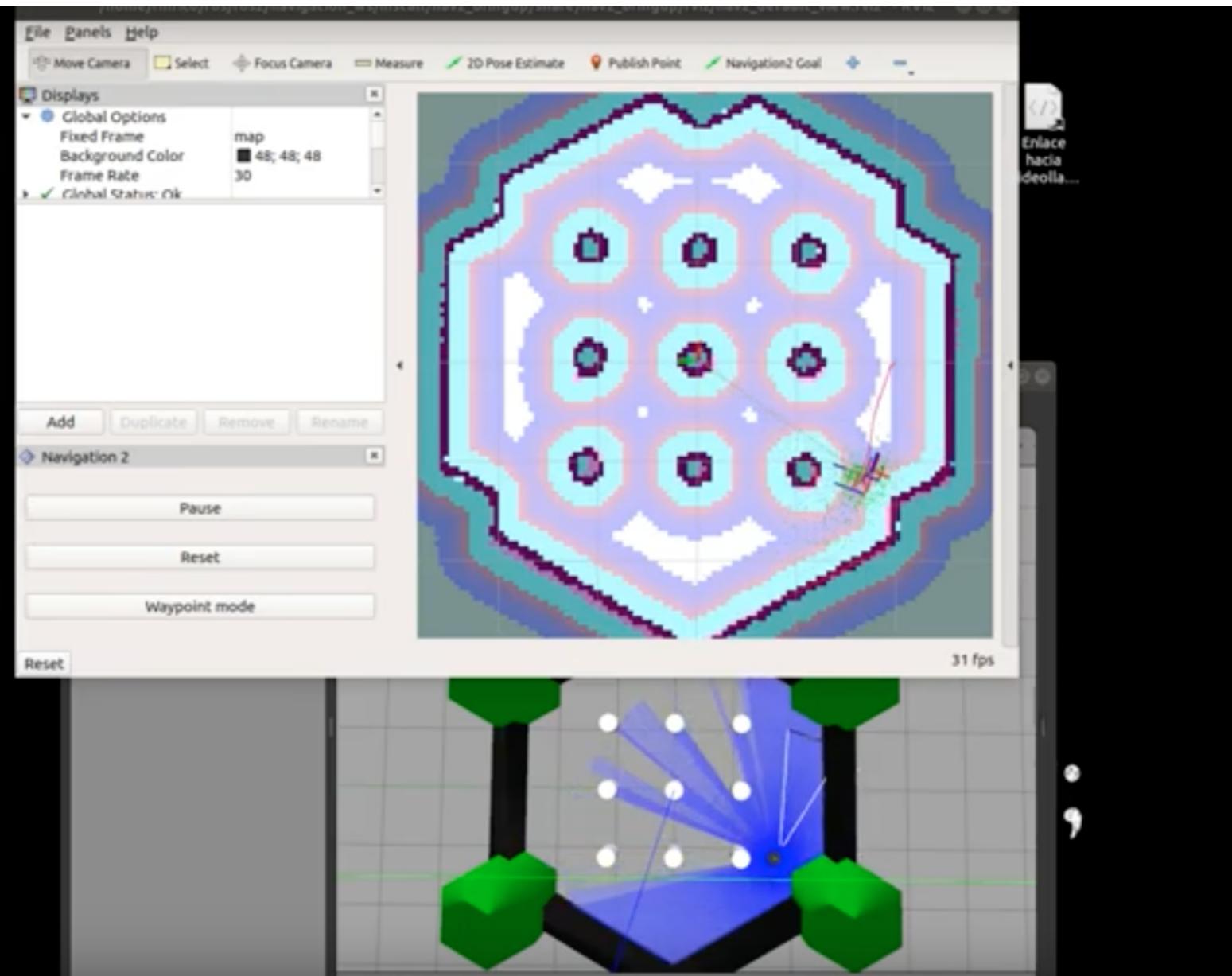
https://github.com/IntelligentRoboticsLabs/ros2_planning_system



PlanSys2



[https://github.com/IntelligentRoboticsLabs/ros2 planning system](https://github.com/IntelligentRoboticsLabs/ros2_planning_system)



2 Planning System



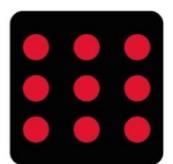
ROS2

for
ROS Developers



Muchas gracias!!!

Encuesta



Intelligent
Robotics
Lab

Prof. Dr. Francisco Martín Rico
Madrid 30-1-2020

 Universidad
Rey Juan Carlos