Robot Operating System (ROS2)

Vilma Muço

Université de Toulon, Master Ingénierie des Systèmes Complexes (ISC) -Erasmus Mundus MIR

November, 2023

Presentation

- Master of Science in Informatics at Grenoble (MoSIG) - Artificial Intelligence and the Web
- Robotics Engineer in Toulon
- working on DriX

UGA Université Grenoble Alpes

exail

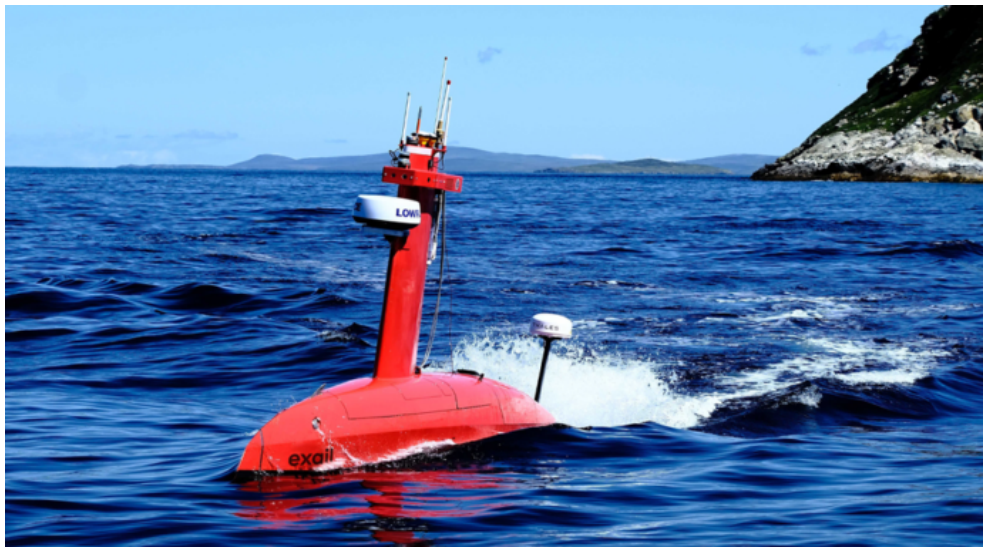


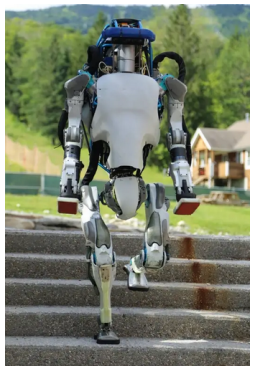
Table of Contents

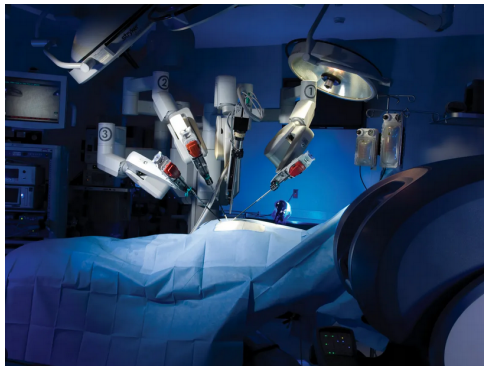
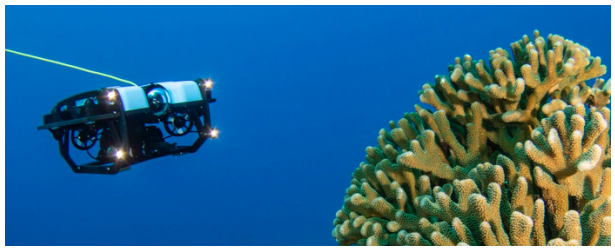
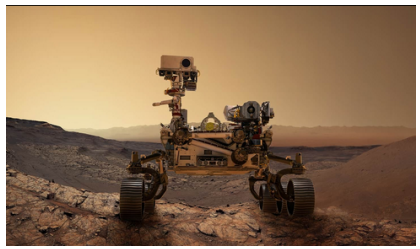
Sources

1. Chapter 1 and 2 of the book Concise Introduction to Robot Programming With Ros2 by Francisco Martín Rico
2. https://docs.ros.org/en/iron/index.html

Middleware for robotics

Robot applications $\rightarrow$ Performing tasks in the real world



Middlewares for Robotics

- Yarp (humanoid robots)
- Carmen
- Player/Stage
- ROS

ROS2 Distributions

Eloquent Elusor	November 22nd, 2019		November 2020
Dashing Diademata	May 31st, 2019		May 2021
Crystal Clemmys	December 14th, 2018		December 2019
Bouncy Bolson	July 2nd, 2018		July 2019
Ardent Apalone	December 8th, 2017		December 2018

Distro	Release date	Logo	EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		December 9th, 2022
Foxy Fitzroy	June 5th, 2020		June 20th, 2023

Latest Distribution

- Iron Irwini
  - Ubuntu Linux - Jammy Jellyfish (22.04)
  - Windows
  - RHEL/Fedora
  - macOS

Upcoming Distribution

- There is a new ROS 2 distribution released yearly on May 23rd (World Turtle Day).
- Jazzy Jalisco on May 2024

Distro	Release date	Logo	EOL date
Jazzy Jalisco	May 2024	TBD	May 2029

- Rolling Ridley

What is ROS?

The Robot Operating System (ROS) is a set of software libraries (state-of-the-art algorithms), methodologies and tools for building robot applications.

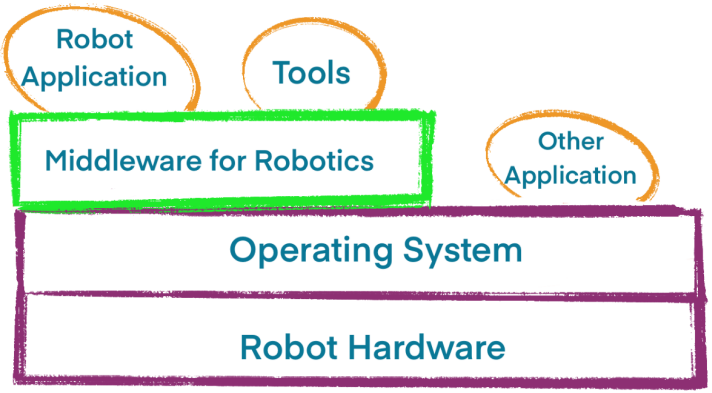


Image adapted from the book *Introduction to Robot Programming With Ros2*

Why ROS?

- Free and Open Source
- **Huge community of highly qualified roboticists**

Why ROS?

- Modular Architecture
- Allows to quickly build and easily connect the main robotic components:
  - Actuators
  - Sensors
  - Control Systems
- microRos for real time OS

Why ROS?

- Tools for monitoring
  - Logs for testing
  - Training
  - Quality Assurance
- Simulated environment and Visualisation Tools

Planes of study

- The Community
- The workspace (static/ development time)
- The computational Graph (dynamic/ runtime)

ROS Community

- Public repositories
- Development methodology
- Software delivery mechanisms
- Open Robotics
  - https://www.ros.org
  - https://docs.ros.org/en/iron/index.html
  - https://answers.ros.org/questions/
- ROSCon
  - Annual ROS developer conference
  - Regional ROS events like ROSConJP and ROSConFr

Workspaces

Combining workspaces using the shell environment

- Advantages:
  - different distros
  - different vesions of packages

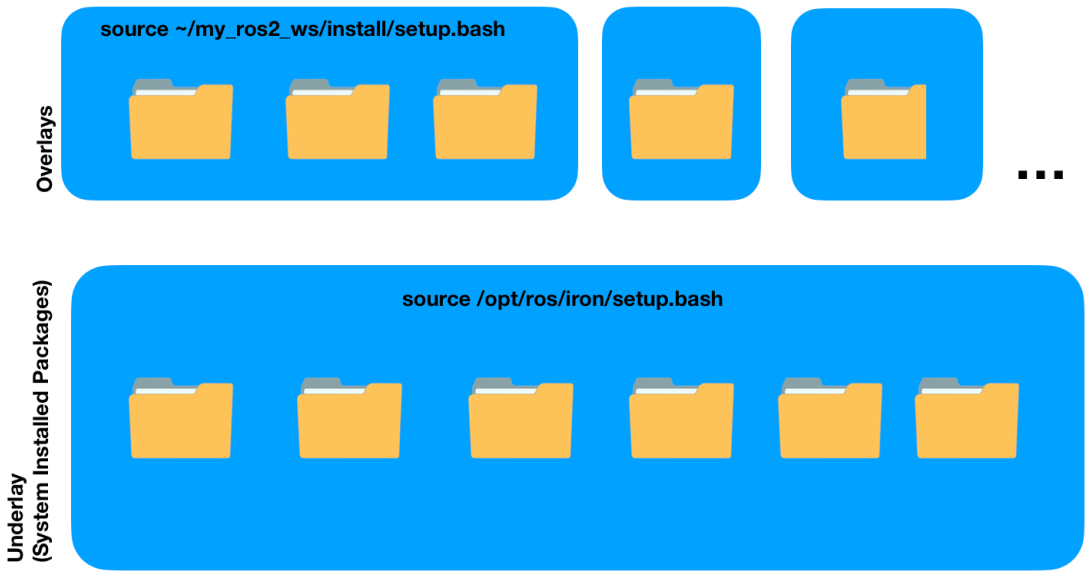
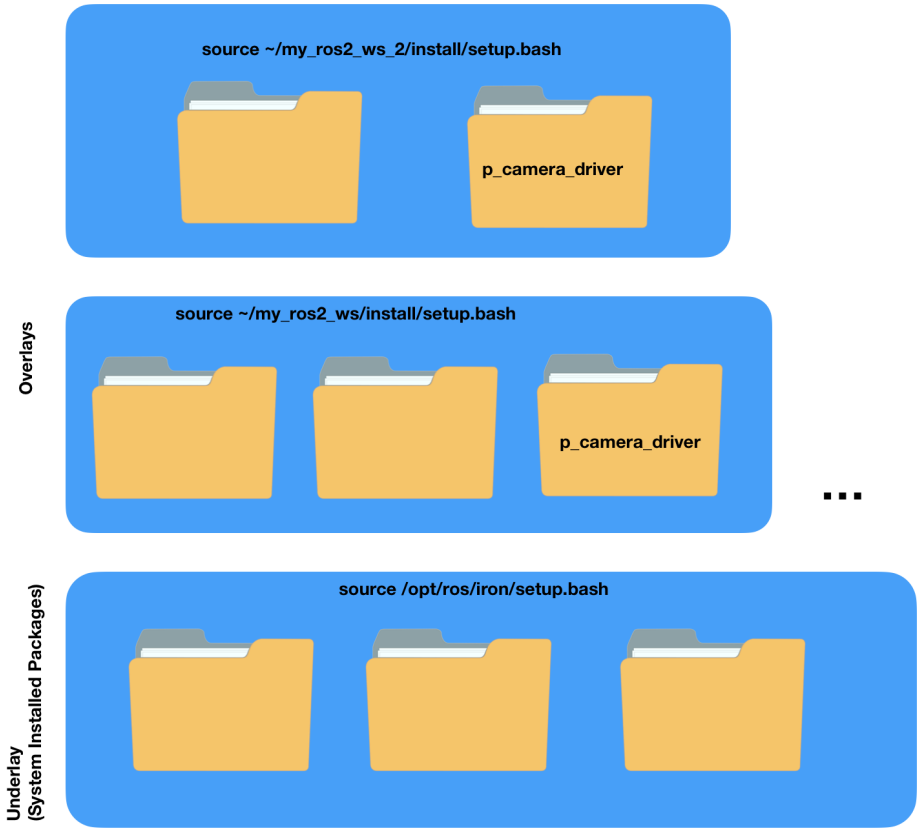


Image adapted from the book *Introduction to Robot Programming With Ros2*

Example Workspaces



ROS Graph

- The computational Graph (dynamic/ runtime)
  - Nodes
  - Links:
    - Topics
    - Services
    - Actions

Nodes

- Fundamental ROS element that is responsible for a single, modular purpose in a robotic system.
- Each node sends and receives data from other nodes
- A single executable (ex. python file) can contain one or more nodes

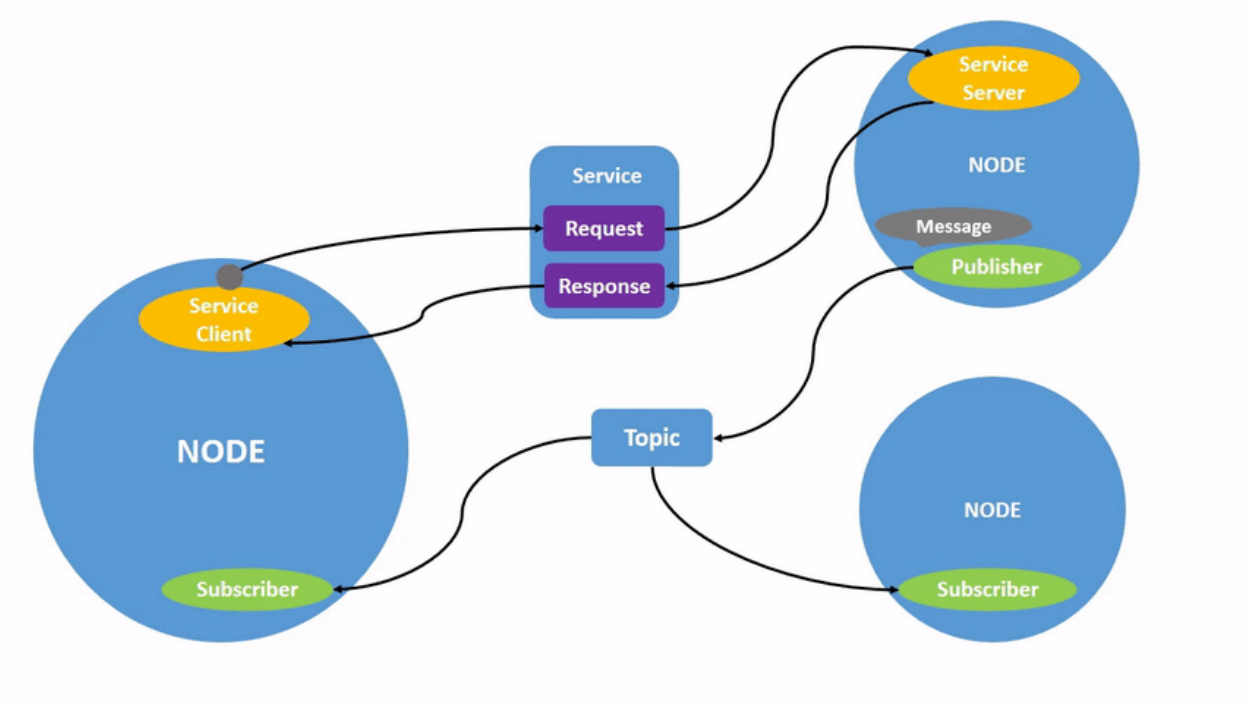


Image taken from *docs.ros.org*

- Asynchronous way of transporting messages
- Subscriber and Publisher

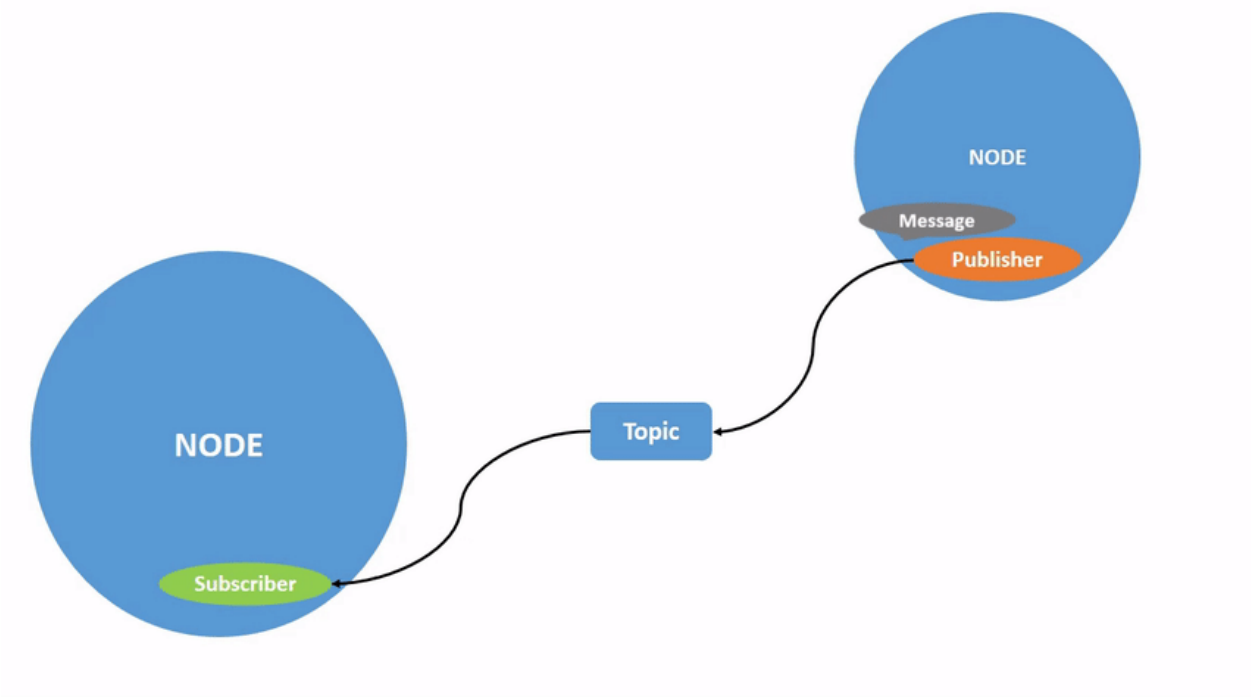


Image taken from *docs.ros.org*

- A node may publish data to any number of topics and simultaneously have subscriptions to any number of topics.

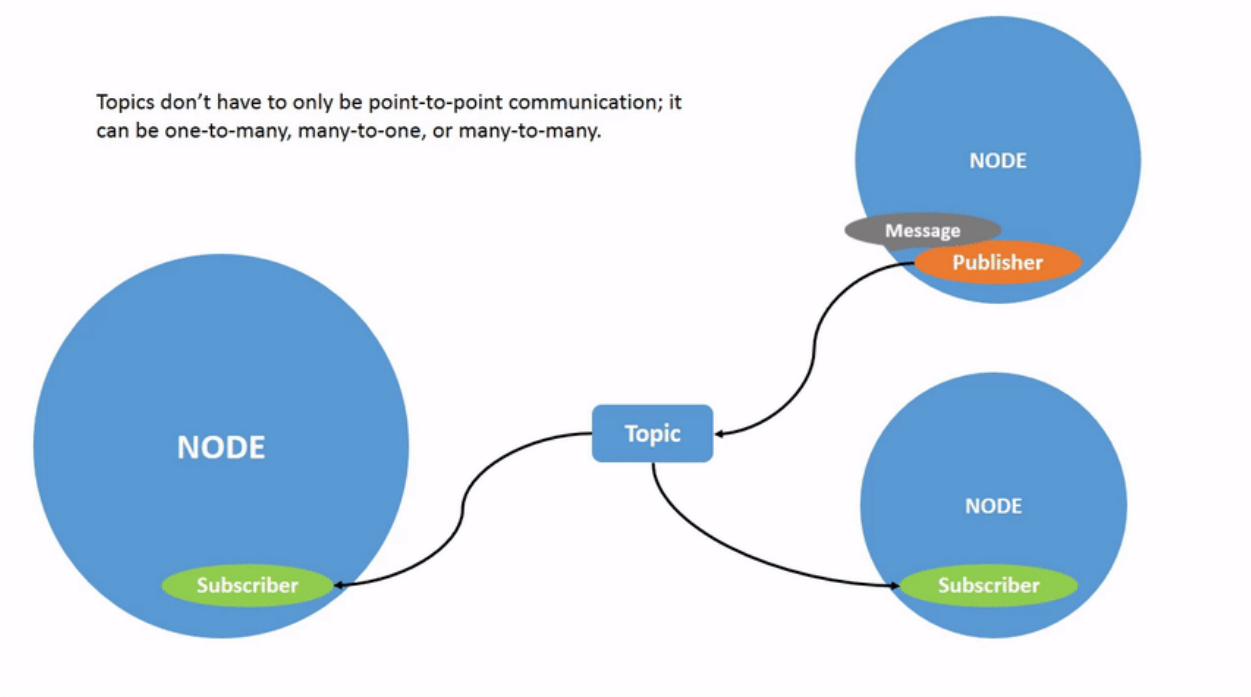
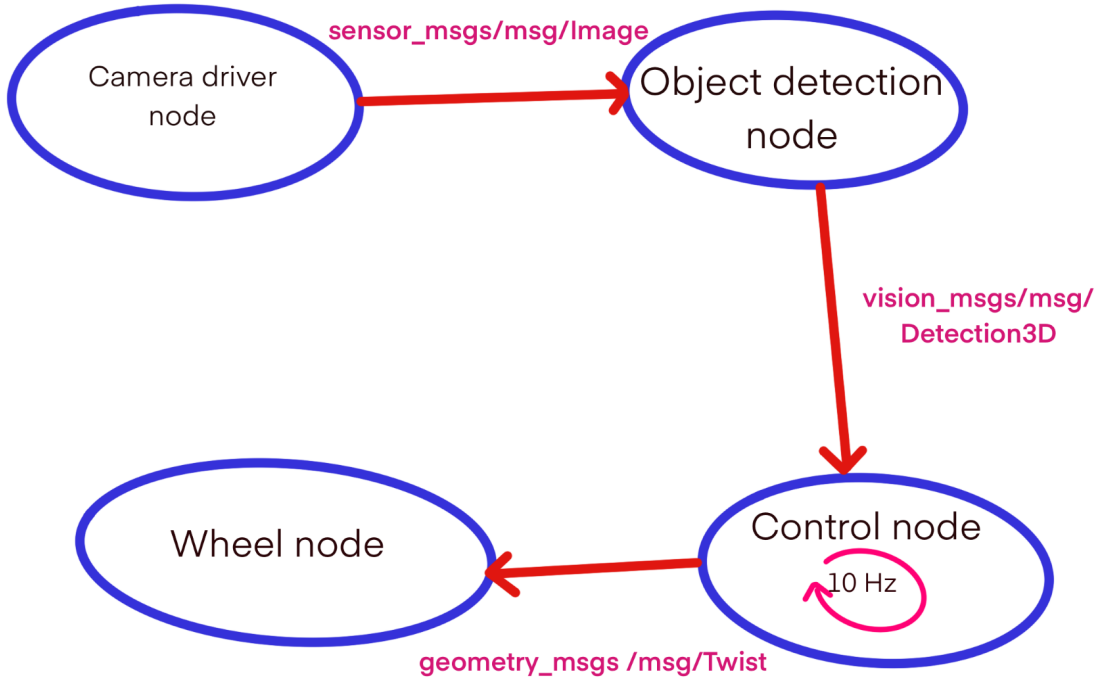


Image taken from *docs.ros.org*

In the example notice:

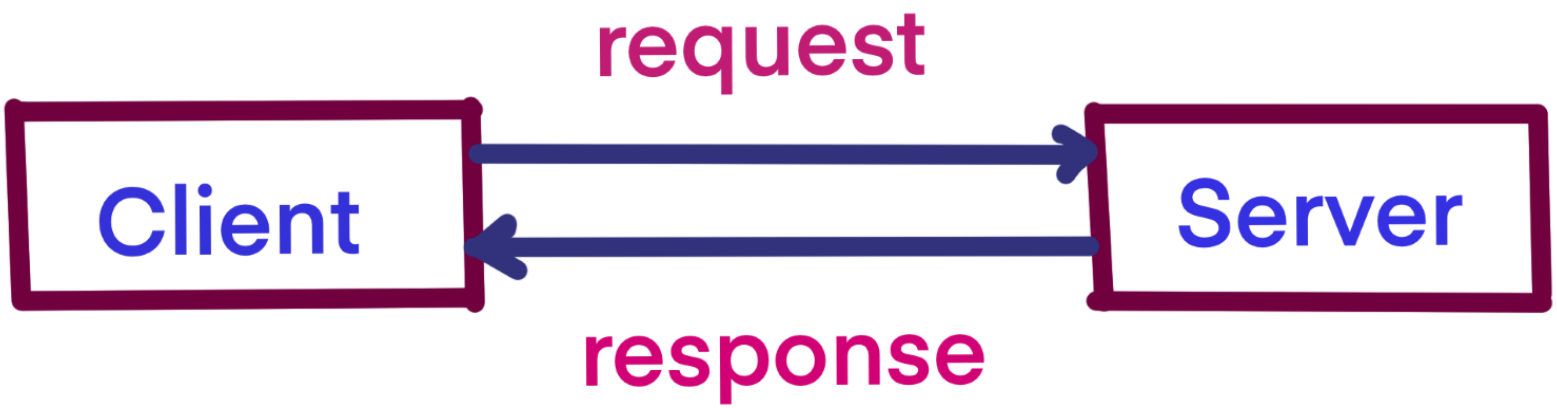
- Node execution modes
  - Iterative execution
  - Event-oriented execution
- Message types



Messages in ROS

- The names of the resources in ROS follow a convention very similar to the file system in Unix (ex. /turtle1/cmd_vel)
- namespaces to isolate resources
- predefined message types (geometry_msgs/msg/Twist - virtually all robots in ROS2 receive these types of messages to control their speed)
- We can define our own message types in ROS

- Call-and-response model
- Services only provide data when they are specifically called by a client (Synchronous way of communication).



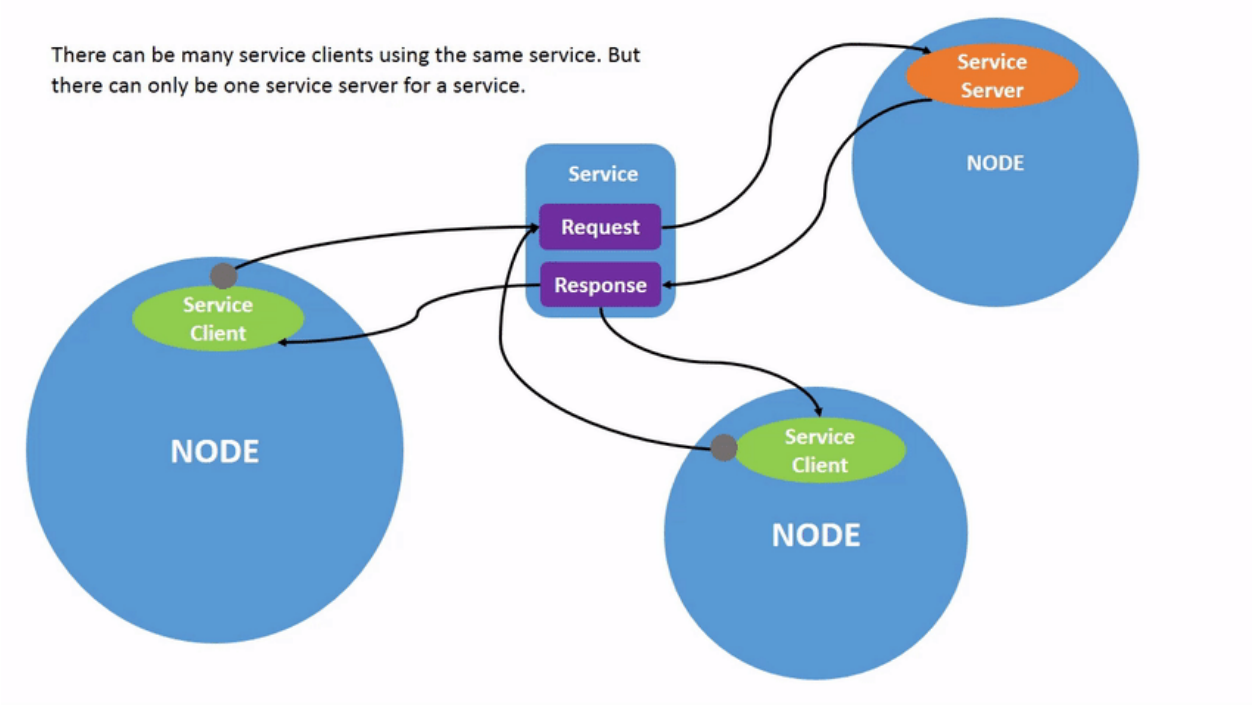
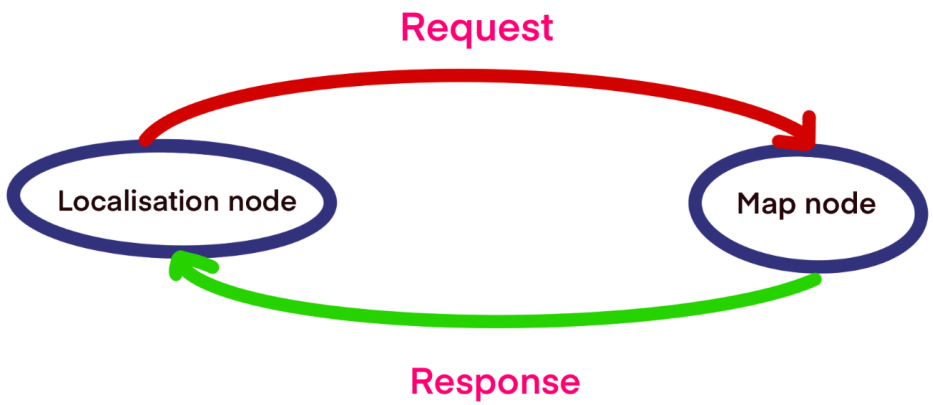


Image taken from *docs.ros.org*

- The call to reset a map with a response if the call succeeded



- Actions are like services that allow you to execute long running tasks (like navigation), provide regular feedback, and can be canceled.
- They consist of three parts:
  - A goal
  - Feedback
  - A result
- Actions are built on topics and services.
- An “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result.

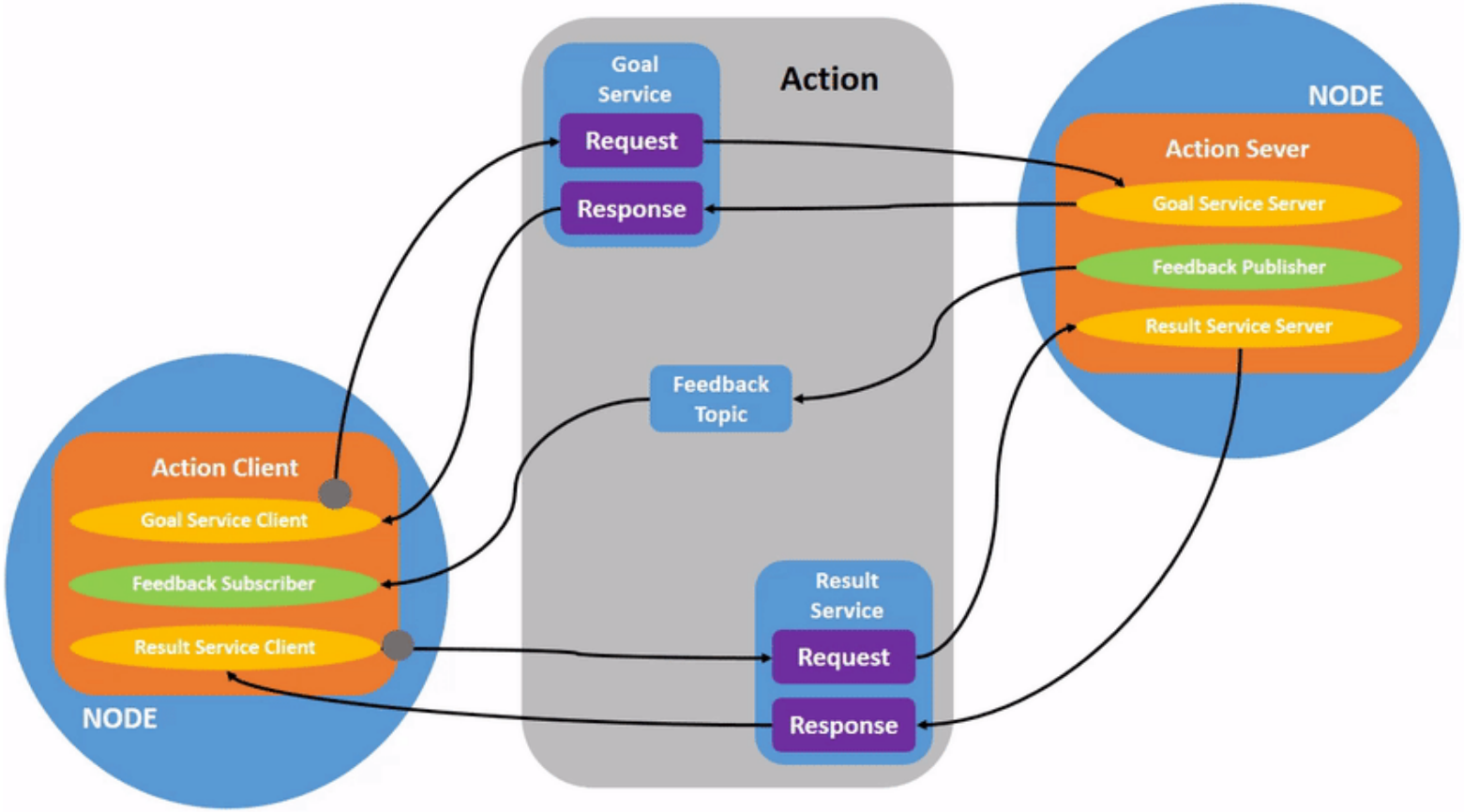
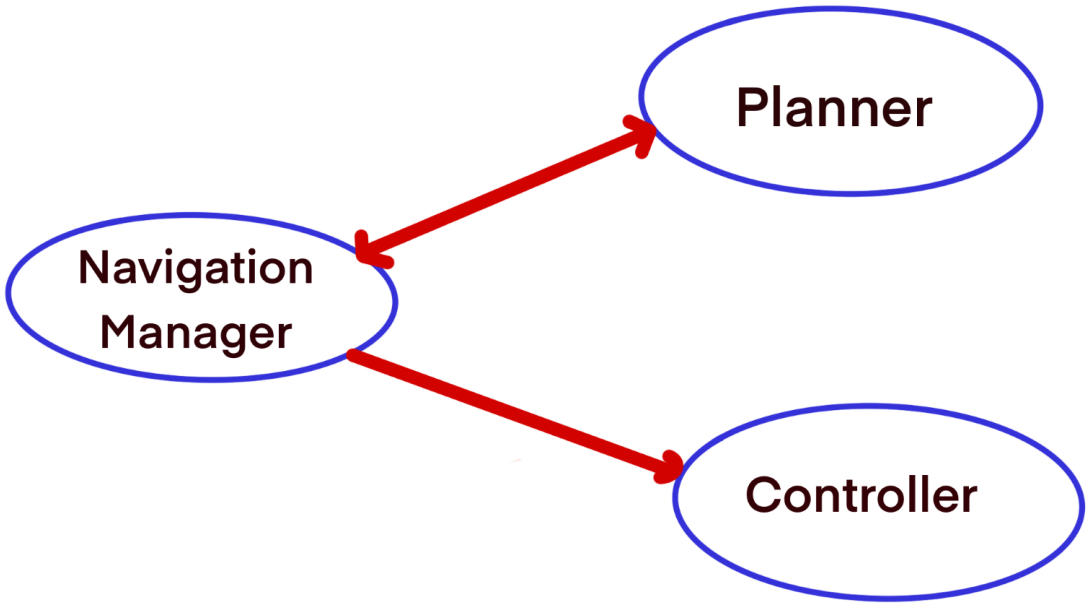


Image taken from *docs.ros.org*

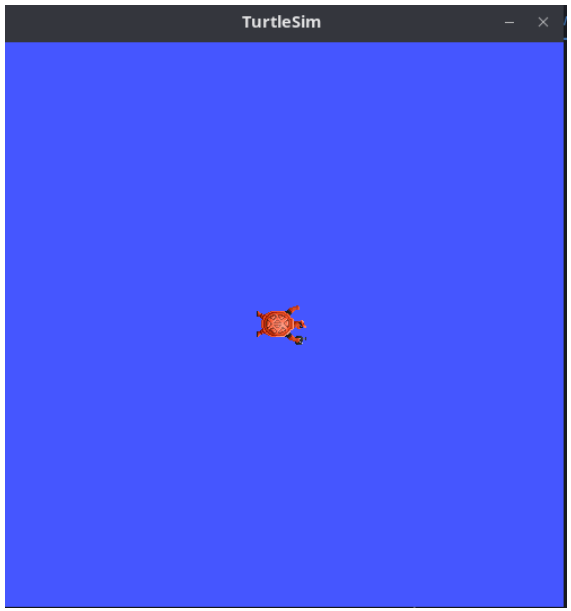
- A navigation request



Turtlesim and the ros command line tool

- A lightweight simulator for learning ROS2
- The ros2 command line tool is how the user manages, introspects, and interacts with a ROS System.

```
$ sudo apt install ros-iron-turtlesim
$ ros2 run turtlesim turtlesim_node
```



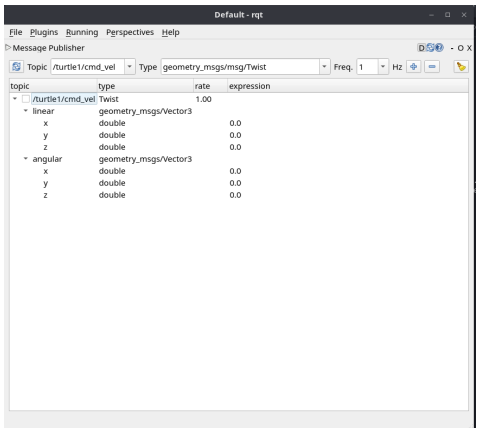
Ros command line tool

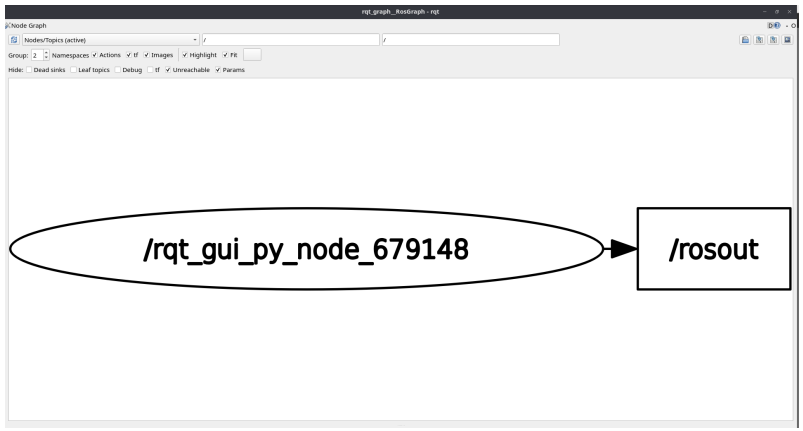
```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 node list
/turtlesim
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 0
Subscription count: 1
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 interface show geometry_msgs/msg/Twist
# This expresses velocity in free space broken into its linear and angular parts.

Vector3  linear
        float64 x
        float64 y
        float64 z
Vector3  angular
        float64 x
        float64 y
        float64 z
vilma@vilma-Precision-3571:~/ros2_ws$
```

rqt

- rqt is a graphical user interface (GUI) tool for ROS 2
- Everything done in rqt can be done on the command line

```
$ sudo apt update
$ sudo apt install ~nros-iron-rqt*
$ rqt
$ rqt_graph
```





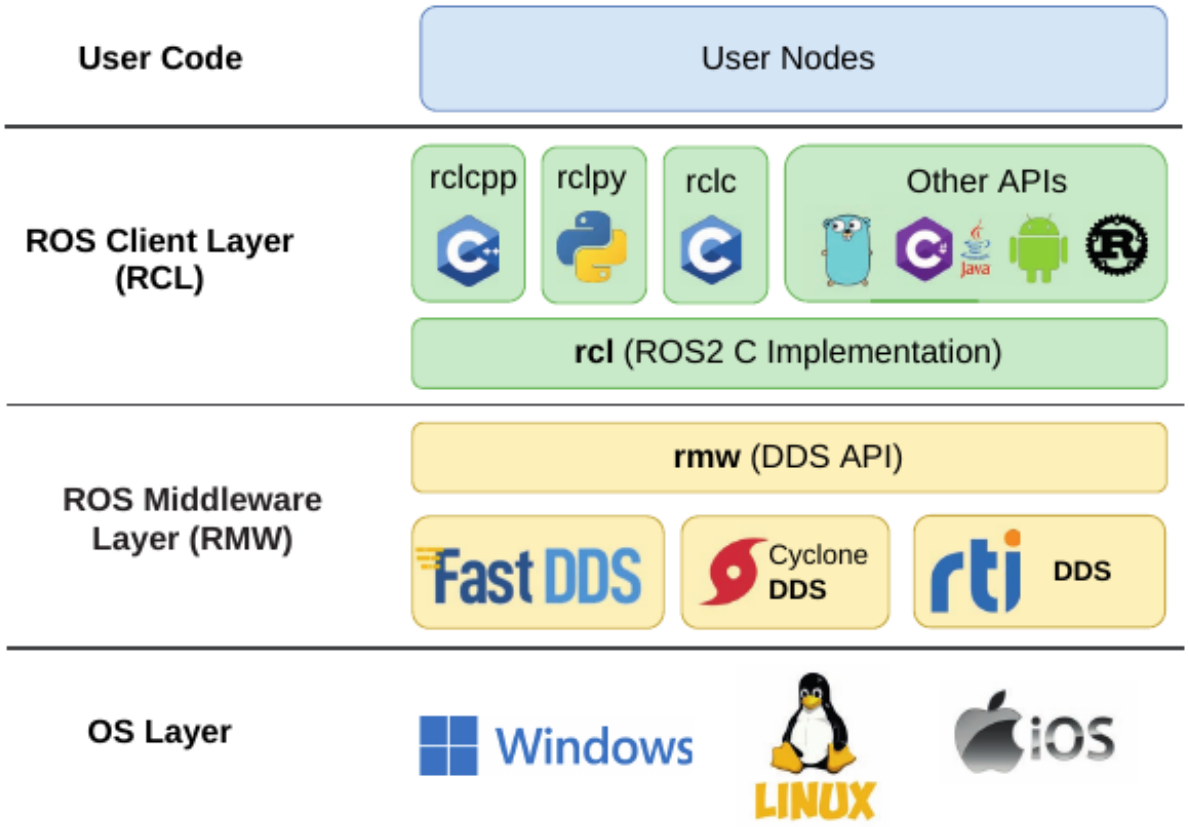
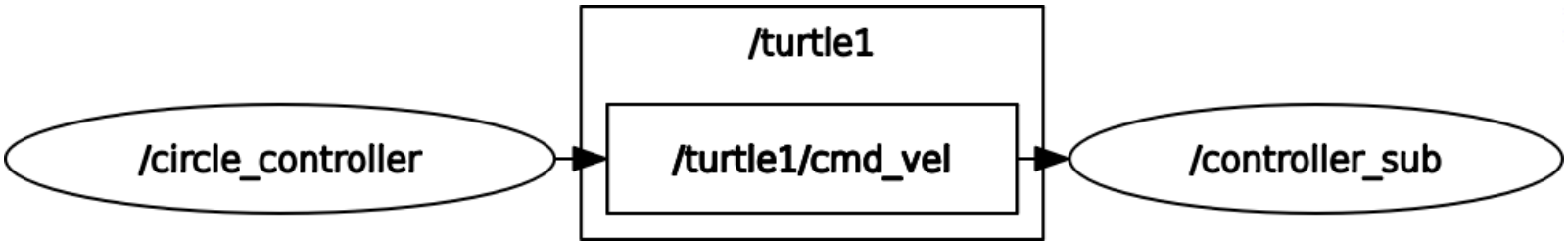


Image taken from the book *Introduction to Robot Programming With Ros2*

- The goal is to send velocity commands in order to move in a circle



- A package contains executables, libraries, or message definitions with a common purpose.
- colcon = collective contruction is a command line tool for building, testing, and using multiple software packages.

- Source the setup file or add the command to your shell startup script:

```
$ source /opt/ros/iron/setup.bash
$ echo "source /opt/ros/iron/setup.bash" >> ~/.bashrc
```

- Create a workspace and a package

```
1 $ mkdir -p ~/ros2_ws/src
2 $ cd ~/ros2_ws/src
3 $ ros2 pkg create --build-type ament_python --node-name \
4   circle_controller  p_controller --license Apache-2.0
5 $ colcon build --packages-select p_controller
6 $ source install/setup.bash
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ cd p_controller/
vilma@vilma-Precision-3571:~/ros2_ws/p_controller$ ll
total 32
drwxrwxr-x 5 vilma vilma 4096 nov.   5 20:26 ./
drwxrwxr-x 7 vilma vilma 4096 nov.   5 20:26 ../
-rw-rw-r-- 1 vilma vilma  633 nov.   5 20:26 package.xml
drwxrwxr-x 2 vilma vilma 4096 nov.   5 20:26 p_controller/
drwxrwxr-x 2 vilma vilma 4096 nov.   5 20:26 resource/
-rw-rw-r-- 1 vilma vilma   93 nov.   5 20:26 setup.cfg
-rw-rw-r-- 1 vilma vilma  725 nov.   5 20:26 setup.py
drwxrwxr-x 2 vilma vilma 4096 nov.   5 20:26 test/
vilma@vilma-Precision-3571:~/ros2_ws/p_controller$
```

- package.xml - file containing meta information about the package
- package_name - a directory with the same name as your package, used by ROS 2 tools to find your package, contains __init__.py
- resource/package_name marker file for the package
- setup.cfg - required when a package has executables, so ros2 run can find them
- setup.py - containing instructions for how to install the package

Customize package.xml

- Description, License, Maintainer

```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>p_controller</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="vilma.muco@exail.com">vilma</maintainer>
  <license>TODO: License declaration</license>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

Customize setup.py

- Same description, maintainer and license fields as package.xml
- The version and name (package_name) also need to **match exactly**

```
from setuptools import find_packages, setup

package_name = 'p_controller'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='vilma',
    maintainer_email='vilma.muco@exail.com',
    description='TODO: Package description',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'circle_controller = p_controller.circle_controller:main'
        ],
    },
)
```

Publisher python code

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist


class CircleController(Node):
    def __init__(self):
        super().__init__('circle_controller')
        self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
        timer_period = 0.1  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        vel = Twist()
        vel.linear.x = 2.0  # linear velocity = radius * angular velocity
        vel.linear.y = 0.0
        vel.linear.z = 0.0
        vel.angular.x = 0.0
        vel.angular.y = 0.0
        vel.angular.z = 1.0
        self.publisher_.publish(vel)


def main(args=None):
    rclpy.init(args=args)

    circleController = CircleController()

    rclpy.spin(circleController)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    circleController.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()

```

Code Analysis

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist


class CircleController(Node):
    def __init__(self):
        super().__init__('circle_controller')
        self.publisher_ = self.create_publisher(Twist, 'turtle1/cmd_vel', 10)
        timer_period = 0.1  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)

    def timer_callback(self):
        vel = Twist()
        vel.linear.x = 2.0  # linear velocity = radius * angular velocity
        vel.linear.y = 0.0
        vel.linear.z = 0.0
        vel.angular.x = 0.0
        vel.angular.y = 0.0
        vel.angular.z = 1.0
        self.publisher_.publish(vel)
```

Code Analysis

```
1  def main(args=None):
2      rclpy.init(args=args)
3
4      circleController = CircleController()
5
6      rclpy.spin(circleController)
7
8      # Destroy the node explicitly
9      # (optional - otherwise it will be done automatically
10     # when the garbage collector destroys the node object)
11     circleController.destroy_node()
12     rclpy.shutdown()
```

Code Analysis

- Add dependencies

package.xml

```
1  <?xml version="1.0"?>
2  <?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
3  <package format="3">
4    <name>p_controller</name>
5    <version>0.0.0</version>
6    <description>TODO: Package description</description>
7    <maintainer email="vilma.muco@exail.com">vilma</maintainer>
8    <license>Apache License 2.0</license>
9
10   <test_depend>ament_copyright</test_depend>
11   <test_depend>ament_flake8</test_depend>
12   <test_depend>ament_pep257</test_depend>
13   <test_depend>python3-pytest</test_depend>
14   <exec_depend>rclpy</exec_depend>
15   <exec_depend>std_msgs</exec_depend>
16
17   <export>
18     <build_type>ament_python</build_type>
19   </export>
20 </package>
```

Code Analysis

- Add entry point

setup.py > ...

```python
from setuptools import find_packages, setup

package_name = 'p_controller'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='vilma',
    maintainer_email='vilma.muco@exail.com',
    description='TODO: Package description',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'circle_controller = p_controller.circle_controller:main'
        ],
    },
)
```

Create a subscriber node

- Create the controller_subscriber.py file in the same folder as the circle_controller.py

```python
import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist

class ControllerSub(Node):
    def __init__(self):
        super().__init__('controller_sub')
        self.subscription = self.create_subscription(
            Twist,
            'turtle1/cmd_vel',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('x linear velocity  = "%s"' % msg.linear.x)
        self.get_logger().info('z angular velocity = "%s"' % msg.angular.z)

def main(args=None):
    rclpy.init(args=args)

    controller_sub = ControllerSub()

    rclpy.spin(controller_sub)

    controller_sub.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Subscriber Code Analysis

```
1  import rclpy
2  from rclpy.node import Node
3  from geometry_msgs.msg import Twist
4
5  class ControllerSub(Node):
6      def __init__(self):
7          super().__init__('controller_sub')
8          self.subscription = self.create_subscription(
9              Twist,
10             'turtle1/cmd_vel',
11             self.listener_callback,
12             10)
13
14     def listener_callback(self, msg):
15         self.get_logger().info('x linear velocity  = "%s"' % msg.linear.x)
16         self.get_logger().info('z angular velocity = "%s"' % msg.angular.z)
17
```

Subscriber Code Analysis

```
1  def main(args=None):
2      rclpy.init(args=args)
3      controller_sub = ControllerSub()
4
5      rclpy.spin(controller_sub)
6
7      controller_sub.destroy_node()
8      rclpy.shutdown()
9
10 if __name__ == '__main__':
11     main()
```

Subscriber Code Analysis

- Add dependencies
- Add entry point (line 24)

```
1  from setuptools import find_packages, setup
2
3  package_name = 'p_controller'
4
5  setup(
6      name=package_name,
7      version='0.0.0',
8      packages=find_packages(exclude=['test']),
9      data_files=[
10         ('share/ament_index/resource_index/packages',
11             ['resource/' + package_name]),
12         ('share/' + package_name, ['package.xml']),
13     ],
14     install_requires=['setuptools'],
15     zip_safe=True,
16     maintainer='vilma',
17     maintainer_email='vilma.muco@exail.com',
18     description='TODO: Package description',
19     license='Apache License 2.0',
20     tests_require=['pytest'],
21     entry_points={
22         'console_scripts': [
23             'circle_controller = p_controller.circle_controller:main',
24             'controller_sub = p_controller.controller_subscriber:main',
25         ],
26     },
27 )
```

Build and run

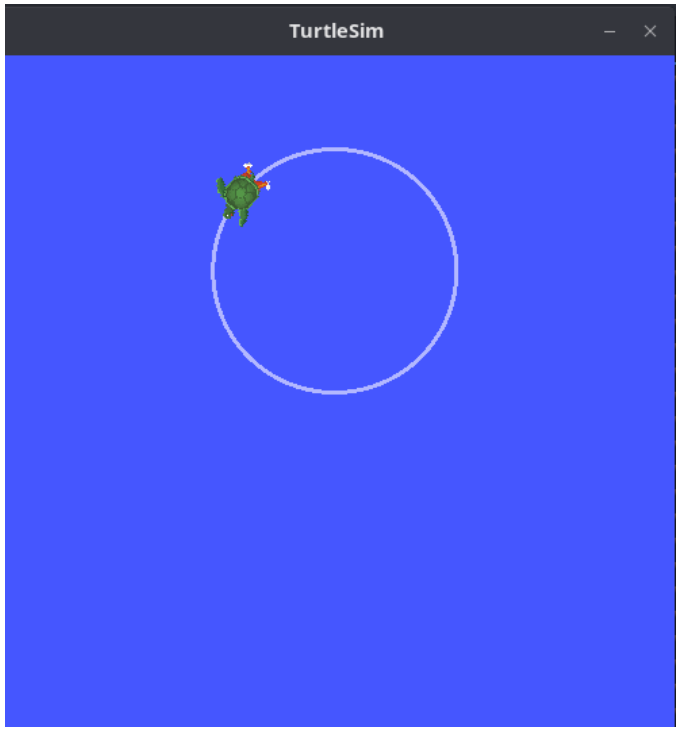
- Run in different terminals the publisher and subscriber

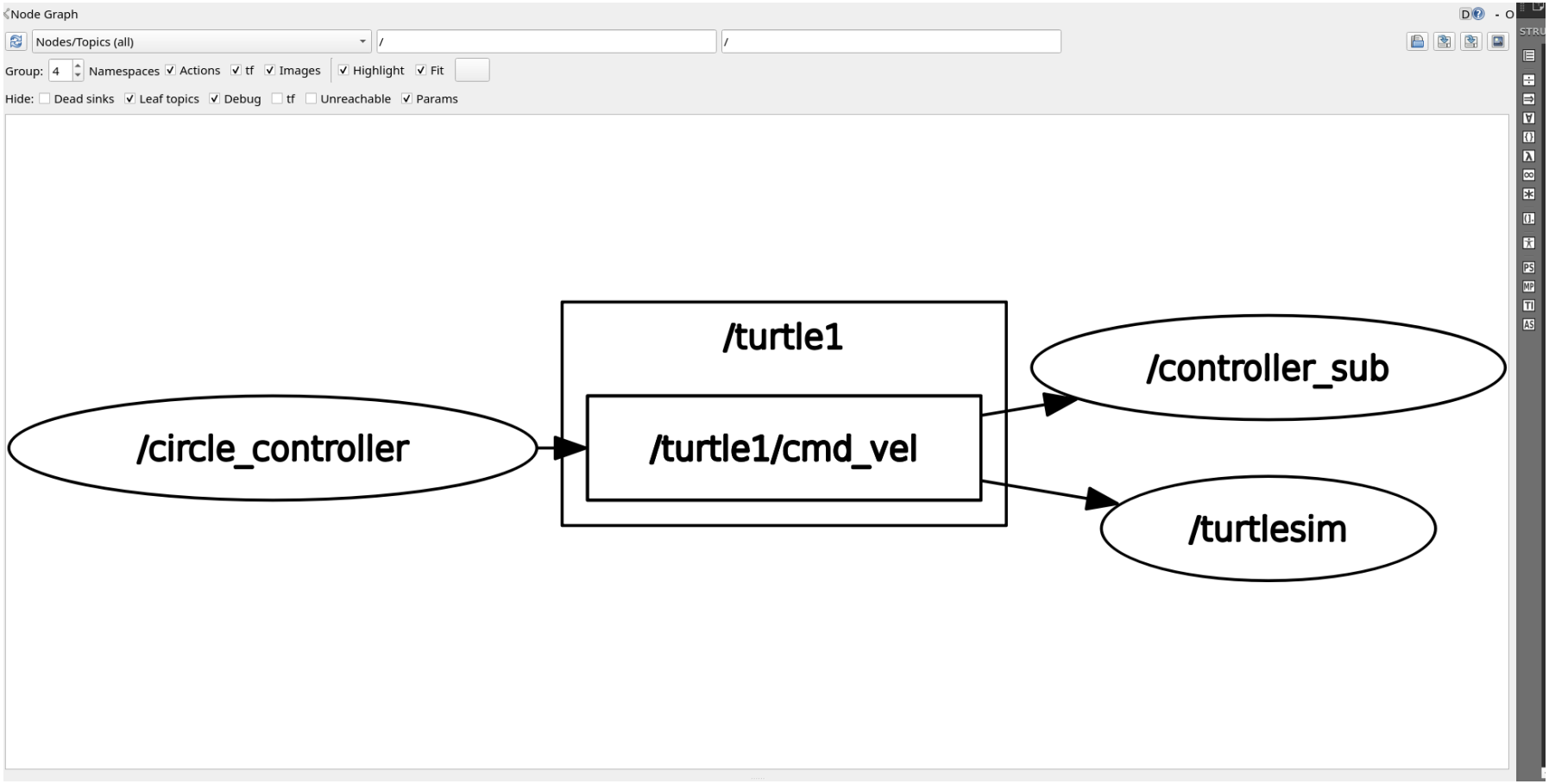
```
1 $ colcon build --packages-select p_controller
2 $ source install/setup.bash
3 $ ros2 run p_controller circle_controller
4 $ ros2 run p_controller controller_sub
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ colcon build --packages-select p_controller
Starting >>> p_controller
Finished <<< p_controller [0.47s]

Summary: 1 package finished [0.56s]
vilma@vilma-Precision-3571:~/ros2_ws$ source install/setup.bash
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 run p_controller circle_controller
```

```
vilma@vilma-Precision-3571:~/ros2_ws$ source install/setup.bash
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 run p_controller controller_sub
[INFO] [1699286368.171228256] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.171394386] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.266327440] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.266923092] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.366044594] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.366566007] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.466493093] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.467171054] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.565943103] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.566630277] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.666639875] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.667474295] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.766729452] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.767555267] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.866700359] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.867304074] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286368.966105988] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286368.966915584] [controller_sub]: z angular velocity = "1.0"
[INFO] [1699286369.066505712] [controller_sub]: x linear velocity  = "2.0"
[INFO] [1699286369.067308922] [controller_sub]: z angular velocity = "1.0"
```

- Run Turtlesim and rqt_graph




Node Graph
Nodes/Topics (all)
Group: 4 Namespaces Actions tf Images Highlight Fit
Hide: Dead sinks Leaf topics Debug tf Unreachable Params
/turtle1
/circle_controller
/turtle1/cmd_vel
/controller_sub
/turtlesim

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic list
/parameter_events
/rosout
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic info /turtle1/cmd_vel
Type: geometry_msgs/msg/Twist
Publisher count: 1
Subscription count: 2
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 topic echo /turtle1/cmd_vel
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
linear:
  x: 2.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 1.0
---
```

- Launch files allow you to start up and configure a number of executables containing ROS 2 nodes simultaneously.
- You can write the files in different languages:
  - Python
  - XML
  - YAML

- A program that contains a LaunchDescription object
- The LaunchDescription object contains actions:
  - Node action - to run a program
  - IncludeLaunchDescription
  - DeclareLaunchArguments
  - SetEnvironmentVariable
  - ...

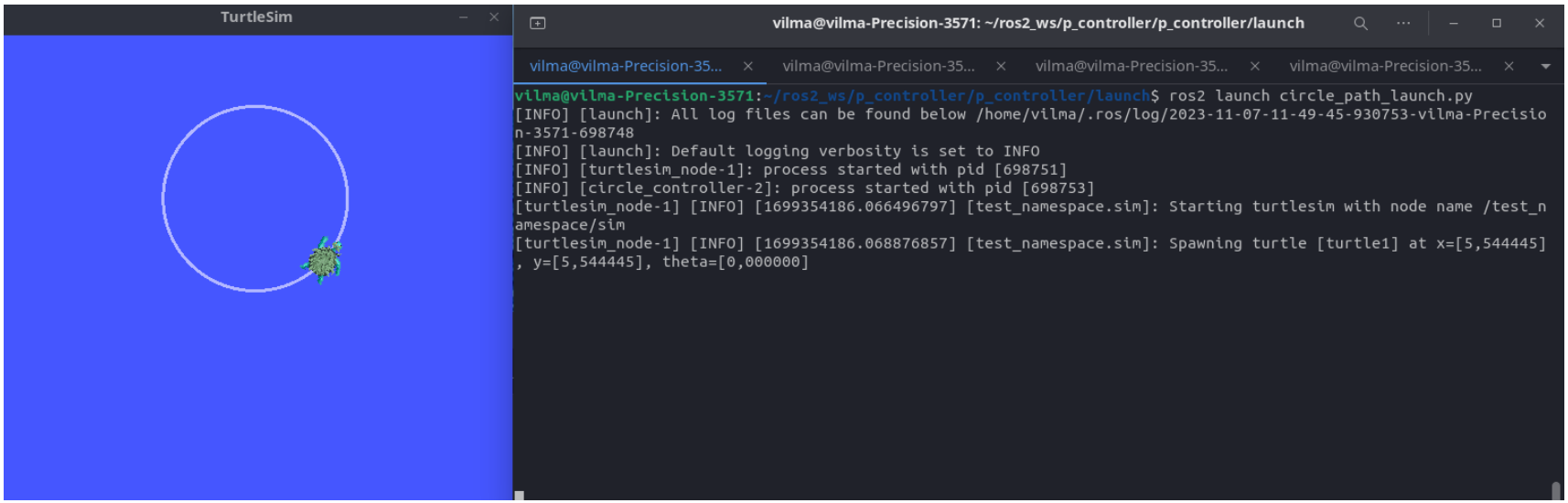
- Create a new directory to store your launch files:

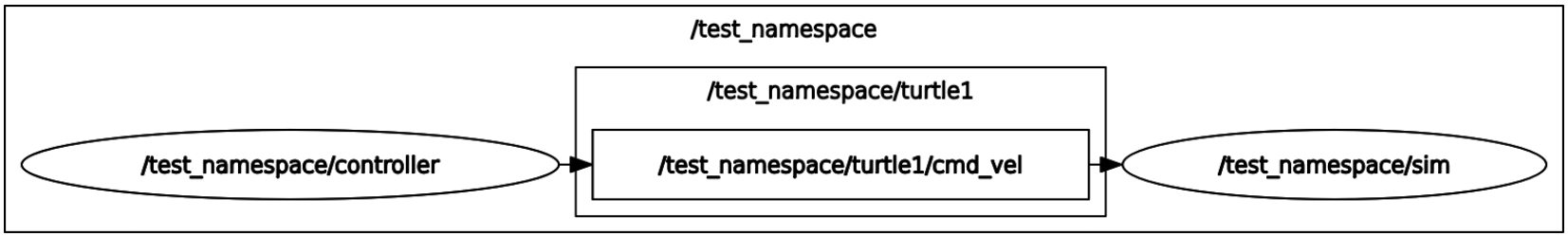
```
$ mkdir launch
```

- Let's put together a ROS 2 launch file using the turtlesim package and the circle_controller publisher.

```python
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='test_namespace',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='p_controller',
            namespace='test_namespace',
            executable='circle_controller',
            name='controller'
        )
    ])

```

```
$ cd launch
$ ros2 launch circle_path_launch.py
```




/test_namespace
/test_namespace/turtle1
/test_namespace/controller
/test_namespace/turtle1/cmd_vel
/test_namespace/sim

- Install Iron Irwini https://docs.ros.org/en/iron/Installation.html
- Write a python publisher node that sends velocity commands to move the turtle in a square (Assume that we do not have any message loss)

Setup instructions and slides:
https://github.com/vilmamuco/ros2-exercises

Thank you!

Questions?