# Robot Operating System (ROS2)

Vilma Muço

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

November, 2023

## Table of Contents

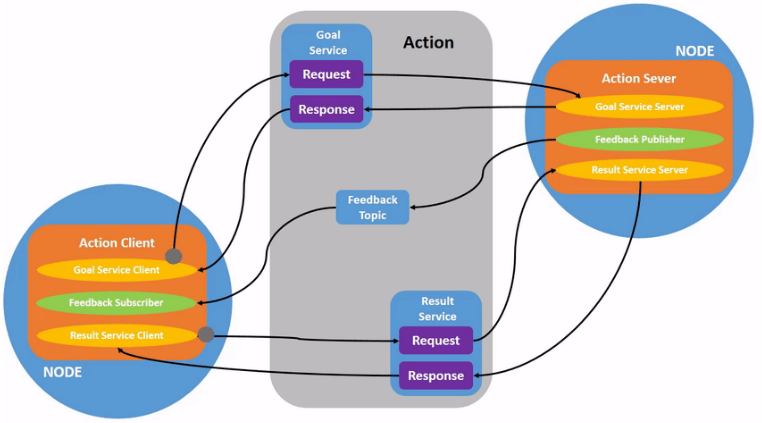


Image taken from *docs.ros.org*

## Custom Actions

- We can custom-define actions in our packages
- Actions are defined in .action files
- A request message is sent from an action client to an action server initiating a new goal.
- A result message is sent from an action server to an action client when a goal is done.
- Feedback messages are periodically sent from an action server to an action client with updates about a goal.

```
# Request
---
# Result
---
# Feedback
```

## Custom Actions

- Create a new package

```
$cd ~/ros2_ws/src
$ros2 pkg create --license Apache-2.0 p_action_interfaces
$cd p_action_interfaces
$mkdir action
$vi GoToPoint.action
```

- Create an action with goal to reach a point and stop

```
# Request
float32 x
float32 y
---
# Result
bool success
---
# Feedback
turtlesim/Pose pose
```

## Build the custom action

- Modify the CMakeLists.txt

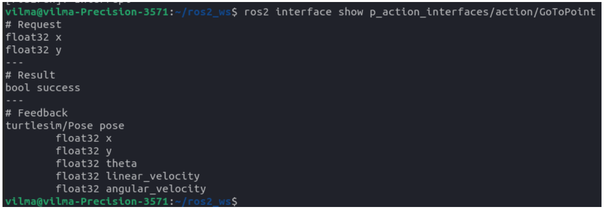
```
find_package(turtlesim REQUIRED)

find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
  "action/GoToPoint.action"
   DEPENDENCIES turtlesim
)
ament_package()
```

- Add the required dependencies to our package.xml

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
<exec_depend>turtlesim</exec_depend>
```

## Build the custom action

```
colcon build --packages-select p_action_interfaces
source install/setup.bash
ros2 interface show p_action_interfaces/action/GoToPoint
```



```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 interface show p_action_interfaces/action/GoToPoint
# Request
float32 x
float32 y
---
# Result
bool success
---
# Feedback
turtlesim/Pose pose
	float32 x
	float32 y
	float32 theta
	float32 linear_velocity
	float32 angular_velocity
vilma@vilma-Precision-3571:~/ros2_ws$
```

## Writing an action server

```python
class GoToGoal(Node):
    def __init__(self):
        super().__init__("go_to_goal")

        self.publisher_ = self.create_publisher(Twist, "turtle1/cmd_vel", 10)

        velocity_callback_group = MutuallyExclusiveCallbackGroup()
        self.subscription = self.create_subscription(
            Pose,
            "turtle1/pose",
            self.pose_listener_callback,
            10,
            callback_group=velocity_callback_group,
        )

        action_callback_group = MutuallyExclusiveCallbackGroup()
        self.action_server = ActionServer(
            self,
            GoToPoint,
            "go_to_point",
            self.action_callback,
            callback_group=action_callback_group,
        )

        self.current_pose = Pose()
        # in the beginning, the turtle is happy where it is until it receives new instructions
        self.goal_reached = True
        self.velocity = Twist()  # 0 by default
        self.goal_x = 0.0
        self.goal_y = 0.0
        self.EPSILON_ERROR = 0.1
```

## Writing an action server

```
    def action_callback(self, goal_handle):
        self.get_logger().info('Executing goal...')
        if not self.goal_reached:
            print('got new goal before old one reached')
        # set the goal
        self.goal_x = goal_handle.request.x
        self.goal_y = goal_handle.request.y
        feedback_msg = GoToPoint.Feedback()
        feedback_msg.pose = self.current_pose

        # while the goal is not reached, send feedback to the client
        diff_x = abs(self.current_pose.x - self.goal_x)
        diff_y = abs(self.current_pose.y - self.goal_y)
        while diff_x > self.EPSILON_ERROR or diff_y > self.EPSILON_ERROR:
            print('sending feedback')
            goal_handle.publish_feedback(feedback_msg)
            feedback_msg.pose = self.current_pose
            diff_x = abs(self.current_pose.x - self.goal_x)
            diff_y = abs(self.current_pose.y - self.goal_y)
            # put only this thread to sleep for 1 seconds
            time.sleep(1)

        goal_handle.succeed()
        result = GoToPoint.Result()
        result.success = True
        self.goal_reached = True
        return result
```

## Writing an action server

```python

def pose_listener_callback(self, msg):
      self.current_pose = msg
      angle = 0.0
      delta = 0.0

      diff_x = abs(self.current_pose.x - self.goal_x)
      diff_y = abs(self.current_pose.y - self.goal_y)
      # calculate the angle between the current pose and the goal pose
      # and rotate the robot towards the goal
      angle = math.atan2(self.goal_y - self.current_pose.y, self.goal_x - self.current_pose.x)

      delta = angle - self.current_pose.theta
      self.velocity.linear.x = 0.0

      distance = math.sqrt(diff_x * diff_x + diff_y * diff_y)
      if diff_x < self.EPSILON_ERROR and diff_y < self.EPSILON_ERROR:
          self.velocity.angular.z = 0.0
      elif delta < -0.02 or delta > 0.02:
          self.velocity.angular.z =  delta * 0.5
      else:
          self.velocity.angular.z = 0.0
          self.velocity.linear.x = distance * 0.5

      # once we have updated what we want to do, publish the velocity
      if self.velocity.linear.x != 0.0 or self.velocity.angular.z != 0.0:
          self.publisher_.publish(self.velocity)
```

## Writing an action server

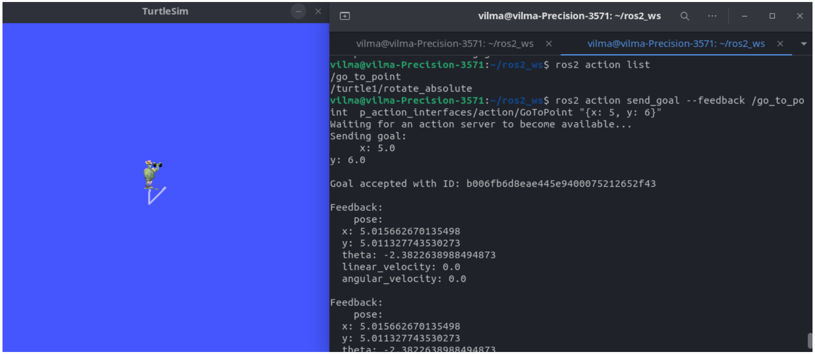
```python
def main(args=None):
    rclpy.init(args=args)

    go_to_goal = GoToGoal()
    executor = MultiThreadedExecutor()  # needed to run functions concurrently
    executor.add_node(go_to_goal)

    executor.spin()

    go_to_goal.destroy_node()
    rclpy.shutdown()


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

## Run the program

- Run turtlesim node and the p_action_example package on two different terminals
- In a third terminal send a goal to the action via command line

```
colcon build --packages-select p_action_example  && ros2 run p_action_example go_to_goal
ros2 run turtlesim turtlesim_node
ros2 action list
ros2 action send_goal --feedback /go_to_point  p_action_interfaces/action/GoToPoint "{x: 5, y: 6}"
```

## Run the program


TurtleSim
vilma@vilma-Precision-3571: ~/ros2_ws
vilma@vilma-Precision-3571: ~/ros2_ws
vilma@vilma-Precision-3571: ~/ros2_ws

```
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 action list
/go_to_point
/turtle1/rotate_absolute
vilma@vilma-Precision-3571:~/ros2_ws$ ros2 action send_goal --feedback /go_to_po
int  p_action_interfaces/action/GoToPoint "{x: 5, y: 6}"
Waiting for an action server to become available...
Sending goal:
     x: 5.0
y: 6.0

Goal accepted with ID: b006fb6d8eae445e9400075212652f43

Feedback:
    pose:
  x: 5.015662670135498
  y: 5.011327743530273
  theta: -2.3822638988494873
  linear_velocity: 0.0
  angular_velocity: 0.0

Feedback:
    pose:
  x: 5.015662670135498
  y: 5.011327743530273
  theta: -2.3822638988494873
```

## Action Client

- Create an action client that moves the turtle in a square

```python
import rclpy
from rclpy.action import ActionClient
from rclpy.node import Node
import time
from p_action_interfaces.action import GoToPoint

class DrawSquareActionClient(Node):
    def __init__(self):
        super().__init__('draw_square_action_client')
        self._action_client = ActionClient(self, GoToPoint, 'go_to_point')
        # create list with 4 edges of the square in tuple format starting from coordinate (5, 5)
        # and with distance of a units between each edge going counter-clockwise
        a = 3.0
        self.edges = [(5.0, 5.0),(a + 5.0, 5.0), (a + 5.0, a + 5.0), (5.0, a + 5.0),(5.0, 5.0)]
        self.current_step = 0 # the current edge of the square the turtle is on

    def send_goal(self):
        x, y = self.edges[self.current_step]
        goal_msg = GoToPoint.Goal()
        goal_msg.x = x
        goal_msg.y = y

        self._action_client.wait_for_server()
```

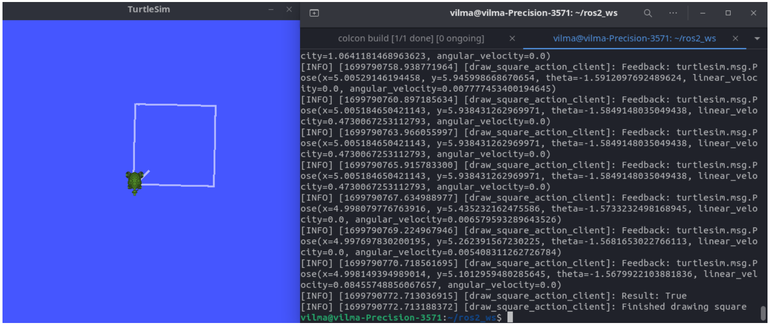
## Action Client

```python
    def send_goal(self):
        x, y = self.edges[self.current_step]
        goal_msg = GoToPoint.Goal()
        goal_msg.x = x
        goal_msg.y = y

        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(goal_msg,
        feedback_callback=self.feedback_callback)
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            return

        self.get_logger().info('Goal accepted :)')
        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)
```

## Action Client

```
    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: {0}'.format(result.success))
        # if result is True go to next step
        if result.success:
            self.current_step += 1
            if self.current_step < len(self.edges):
                self.send_goal()
            else:
                self.get_logger().info('Finished drawing square')
                rclpy.shutdown()

    def feedback_callback(self, feedback_msg):
        self.get_logger().info('Feedback: {0}'.format(feedback_msg.feedback.pose))

def main(args=None):
    rclpy.init(args=args)
    action_client = DrawSquareActionClient()
    action_client.send_goal()
    rclpy.spin(action_client)

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

## Run the program


TurtleSim
vilma@vilma-Precision-3571: ~/ros2_ws
colcon build [1/1 done] [0 ongoing]
vilma@vilma-Precision-3571: ~/ros2_ws

```
city=1.0641181468963623, angular_velocity=0.0)
[INFO] [1699790758.938771964] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.00529146194458, y=5.945998668670654, theta=-1.5912097692489624, linear_veloc
ity=0.0, angular_velocity=0.007777453400194645)
[INFO] [1699790760.897185634] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790763.966055997] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790765.915783300] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=5.005184650421143, y=5.938431262969971, theta=-1.5849148035049438, linear_velo
city=0.4730067253112793, angular_velocity=0.0)
[INFO] [1699790767.634988977] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.998079776763916, y=5.435232162475586, theta=-1.5733232498168945, linear_velo
city=0.0, angular_velocity=0.006579593289643526)
[INFO] [1699790769.224967946] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.997697830200195, y=5.262391567230225, theta=-1.5681653022766113, linear_velo
city=0.0, angular_velocity=0.005408311262726784)
[INFO] [1699790770.718561695] [draw_square_action_client]: Feedback: turtlesim.msg.P
ose(x=4.998149394989014, y=5.1012959480285645, theta=-1.5679922103881836, linear_vel
ocity=0.08455748856067657, angular_velocity=0.0)
[INFO] [1699790772.713036915] [draw_square_action_client]: Result: True
[INFO] [1699790772.713188372] [draw_square_action_client]: Finished drawing square
vilma@vilma-Precision-3571:~/ros2_ws$
```

- Modify the service example code and use the service $/turtle1/set\_pen$ to draw two circles with 2 different colors

Thank you!!!
Questions?