CENG 483 – BEHAVIORAL ROBOTICS

Homework Set #3 Due : 07.11.2023

Name : Cem Tolga MÜNYAS

Student ID : 270206042

2.6 End Chapter Questions

1.

In the context of Robot Operating System (ROS), publisher and subscriber are two essential themes for their communication infrastructure. Substantially, they are used for communication between nodes. In other words, publishers and subscribers are the basic communication mechanism between nodes using topics.

Publishers are the nodes in ROS which produce and broadcast messages on a particular topic. A publisher node creates and sends messages related to a specific topic, any sensor reading, or any data that wants to be shared.

On the other hand, subscribers are the nodes that receives messages from a specific topic. Subscriber nodes basically listen to and subscribe to a specific topic in order to retrieve the information when the corresponding publisher node performs. Additionally, subscriber nodes process the incoming data and takes actions based on that information.

2.

Messages in ROS are used for data communication between different nodes. They define the structure of the data which has been sent between nodes. Messages can involve different kinds of data. Basically, messages carry specific types of information and are used to transmit that information between different nodes in the system of ROS. As we mentioned earlier in the first question, publishers and subscribers use messages to send and receive data which enables them to communicate. Additionally, the message structure is static and only describes the data being communicated.

On the other hand, services enable request-response communication between nodes. A service call is originated by a node, and another node provides the service response to that call. Request is used for containing the data sent to the service provider. Then, response part forms the respond for sending back the data.

Main differences between services and messages are their purpose and how they are used. Messages are for unidirectional data exchange between nodes, while services are for bidirectional request-response interactions.

3.

Nodes communicate with each other using a publish-subscribe mechanism using topics or via request-response communication using services. Nodes can publish messages to topics, subscribe to topics to receive messages, or provide services that can be called by other nodes.

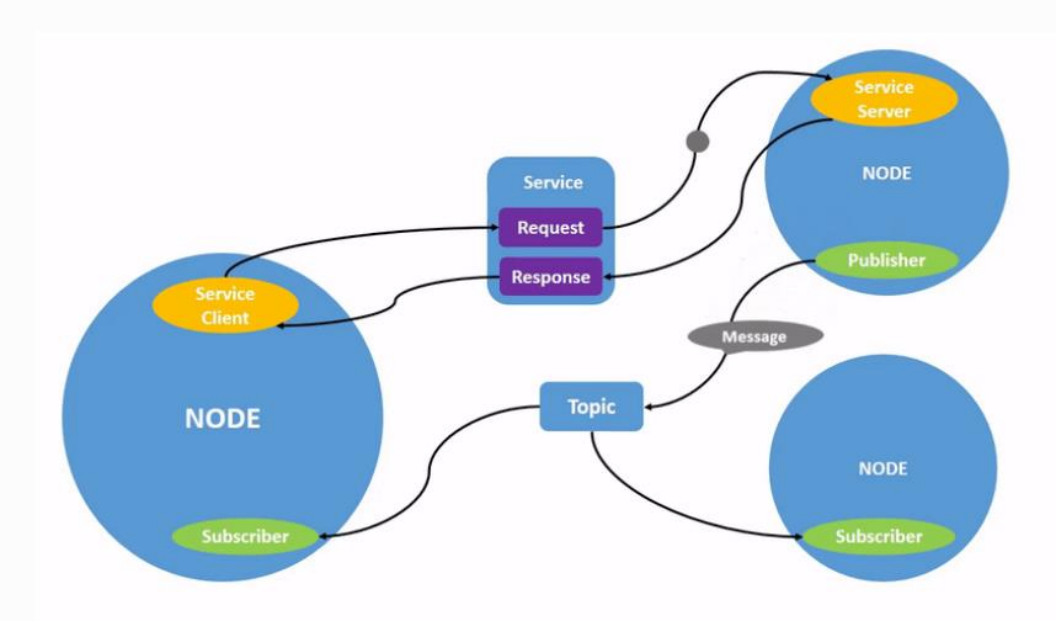

Figure 1.1 Nodes working mechanism

Topics serve as the medium through which nodes can communicate by publishing and subscribing to data. Each topic has a specific name and is associated with a particular type of message.

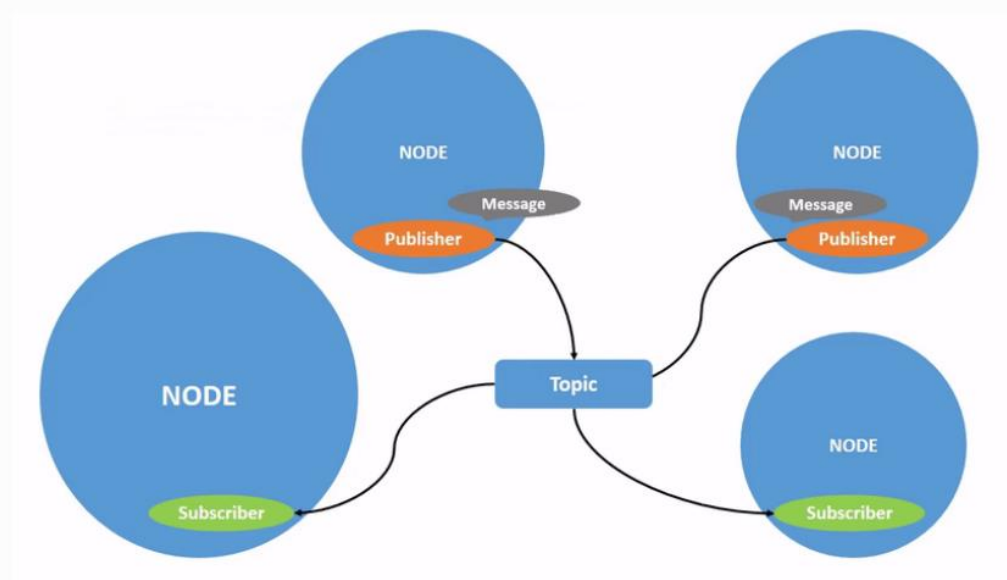

Figure 1.2 Topic working mechanism

Nodes are created using ROS libraries and are responsible for various operations. Nodes are started with a unique name, and they can communicate with other nodes by publishing data to topics, subscribing to topics and using services for request-response interactions.

Topics are used as communication channels where nodes can publish and subscribe to messages.

4.

In order for the nodes to communicate with each other, a service must be created. To do this, first open a new terminal and run ros2. Then, run the following command to create a package :

```
ros2 pkg create --build-type ament_python py_srvcli --dependencies rclpy
example_interfaces
```

This will create a package named `py_srvcli` with all the necessary files and folders.

Next, add the maintainer's email and name, as well as the license information, to the `package.xml` and `setup.py` files.

Now, you can write the service code. Inside the `ros2_ws/src/py_srvcli/py_srvcli` directory, create a new file called `service_member_function.py` and paste the following code:

```python
class AddTwoIntsService(rclpy.node.Node):

    def __init__(self):
        super().__init__('add_two_ints_service')

        # Create a service for adding two integers.
        self.service = self.create_service(
            AddTwoInts,
            'add_two_ints',
            self.handle_add_two_ints)

    def handle_add_two_ints(self, request, response):
        """Handle requests to add two integers."""

        response.sum = request.a + request.b

        return response

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

    node = AddTwoIntsService()

    rclpy.spin(node)

    rclpy.shutdown()

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

Figure 1.3 Service node for adding two integers

This code will create a service node named `add_two_ints_service` that provides a service for adding two integers. The service takes two integers as input and returns the sum of those integers as output.

To use the service, you can create a client node and call the service using the following code:

```python
class AddTwoIntsClient(rclpy.node.Node):
    def __init__(self):
        super().__init__('add_two_ints_client')

        # Create a client for the add_two_ints service.
        self.client = self.create_client(
            AddTwoInts,
            'add_two_ints')

    def send_request(self, a, b):
        """Send a request to add two integers."""
        request = AddTwoInts.Request()
        request.a = a
        request.b = b
        future = self.client.call_async(request)
        future.add_done_callback(self.on_response)

    def on_response(self, future):
        """Handle the response from the add_two_ints service."""
        response = future.result()
        print('The sum is:', response.sum)

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

    node = AddTwoIntsClient()

    # Send a request to add two integers.
    node.send_request(1, 2)
    rclpy.spin(node)
    rclpy.shutdown()

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

Figure 1.4 Calling the service and implementing adding two integers

5.

To use ROS publishers and subscribers with Python, you first need to create a workspace and a package. In this example, we will call the workspace **`ros2_ws`** and the package **`py_pubsub`.**

Once you have created the workspace and package, you need to add the following dependencies to the package.xml file:

```
<dependencies>
<depend>rclpy</depend>
<depend>std_msgs</depend>
</dependencies>
```

You also need to add an entry point to the setup.py file. This entry point will tell ROS where to find the publisher and subscriber code.

```
entry_points={
  'console_scripts': [
    'talker = py_pubsub.publisher_member_function:main',
    'listener = py_pubsub.subscriber_member_function:main',
  ],
}
```

Finally, you need to build the package with colcon and run the publisher and subscriber nodes.

```
colcon build --packages-select py_pubsub
ros2 run py_pubsub talker
ros2 run py_pubsub listener
```

The output in the terminal should be like this :

```
[INFO] [minimal_publisher]: Publishing: "Hello World: 0"
[INFO] [minimal_subscriber]: I heard: "Hello World: 0"
[INFO] [minimal_publisher]: Publishing: "Hello World: 1"
[INFO] [minimal_subscriber]: I heard: "Hello World: 1"
[INFO] [minimal_publisher]: Publishing: "Hello World: 2"
[INFO] [minimal_subscriber]: I heard: "Hello World: 2"
[INFO] [minimal_publisher]: Publishing: "Hello World: 3"
[INFO] [minimal_subscriber]: I heard: "Hello World: 3"
[INFO] [minimal_publisher]: Publishing: "Hello World: 4"
[INFO] [minimal_subscriber]: I heard: "Hello World: 4"
```

Figure 1.5 Terminal output for publisher-subscriber nodes

6.

In order to give a speed command to a robot using Python, we would publish a message of type ***"geometry_msgs/Twist"*** to control the robot's linear and angular velocities. This message is generally published to the ***"/cmd_vel"*** topic.

We need to create a node responsible for publishing the speed command messages in order to publish speed commands to control the robot's motion.

Here is the example Python script for creating a node that published speed commands to control the robot's motion.

```python
def robot_speed_command():
    rospy.init_node('robot_speed_node', anonymous=True)  # Implement the node

    # Create a publisher that publishes messages of type Twist to the /cmd_vel topic
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)

    # Create a Twist message to control linear and angular velocities
    speed_msg = Twist()
    speed_msg.linear.x = 1  # Defining its velocity as 1 m/s
    speed_msg.angular.z = 0.2  # Defining its angular velocity as 0.5 rad/s

    freq = rospy.Rate(20)  # Setting the publishing rate as 20 Hz
    while not rospy.is_shutdown():
        pub.publish(speed_msg)  # Publishing the speed command message
        freq.sleep()


if __name__ == '__main__':
    try:
        robot_speed_command()  # Calling the function to publish speed commands
    except rospy.ROSInterruptException:
        pass
```

Figure 1.3 Python script for giving speed command to a robot