ROBOTICS LAB 7

Parameter Server and Services

MUNADI SIAL

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

Overview

This lab will involve the following:

- Parameters in ROS
- Create a Service Definition
- Implement a Server Node for service response
- Implement a Client Node for service requests

Parameters in ROS

- A parameter is a configuration of a node; it is a variable in the node which can be set by the user when executing the node
- A node can store parameters such as integers, floats, booleans, strings, lists and so on
- In the teleoperation program, the magnitude of the linear velocity and the magnitude of the angular velocity can be 2 parameters. By setting these parameters in the terminal, the user can modify the speed values without having to edit the node script

Parameters in ROS

- Consider a talking robot which uses text-to-speech conversion
- A node can be used to subscribe to a ‘text’ topic, convert the incoming text into audio speech and then publish the audio to a ‘sound’ topic
- When running this node, we may have to change various settings such as the voice pitch, speech rate and audio volume etc
- These settings are all good candidates for parameters which the user can set when starting the node
- Parameters can be thought of as knobs which the user is expected to adjust easily or frequently without modifying the code

Parameters in ROS

The **declare_parameter** function must be used on the node constructor to initialize a parameter. The following statement will create a parameter called 'volume' with a default value of 0.4:

```
self.declare_parameter('volume', 0.4)
```

The **get_parameter** function is used in the callbacks to obtain the value of the parameter. The following statement stores the 'volume' parameter in a variable which can then be used in the code:

```
vol = self.get_parameter('volume').value
```

Parameters in ROS

- In the terminal, the parameter value can be changed by the user:

```
ros2 param set /speech_node volume 0.2
```

- If the above command is not given, then the parameter will simply take the default value in the node

Communication Types

There are 3 ways by which nodes can communicate in ROS:

1 – Topics: The most common method of communicating. A topic is a channel through which messages are passed in a continuous manner. A node publishes a message to a topic. Nodes that have subscribed to the topic can then receive messages

2 – Services: A two-way communication which uses client/server mechanism. A client node sends a request for the service. The server node then responds by executing its callback function. Services are used for short-term communication when messages are not required to be transmitted constantly such as in enabling/disabling motors.

3 – Actions: Similar to services but more flexible. A client node initiates a goal at which the server node responds by executing its callback function. Unlike services, actions provide regular feedback over the status and can also be aborted if the need arises. Actions are used for long–term communication or when the required time is not known.

Services

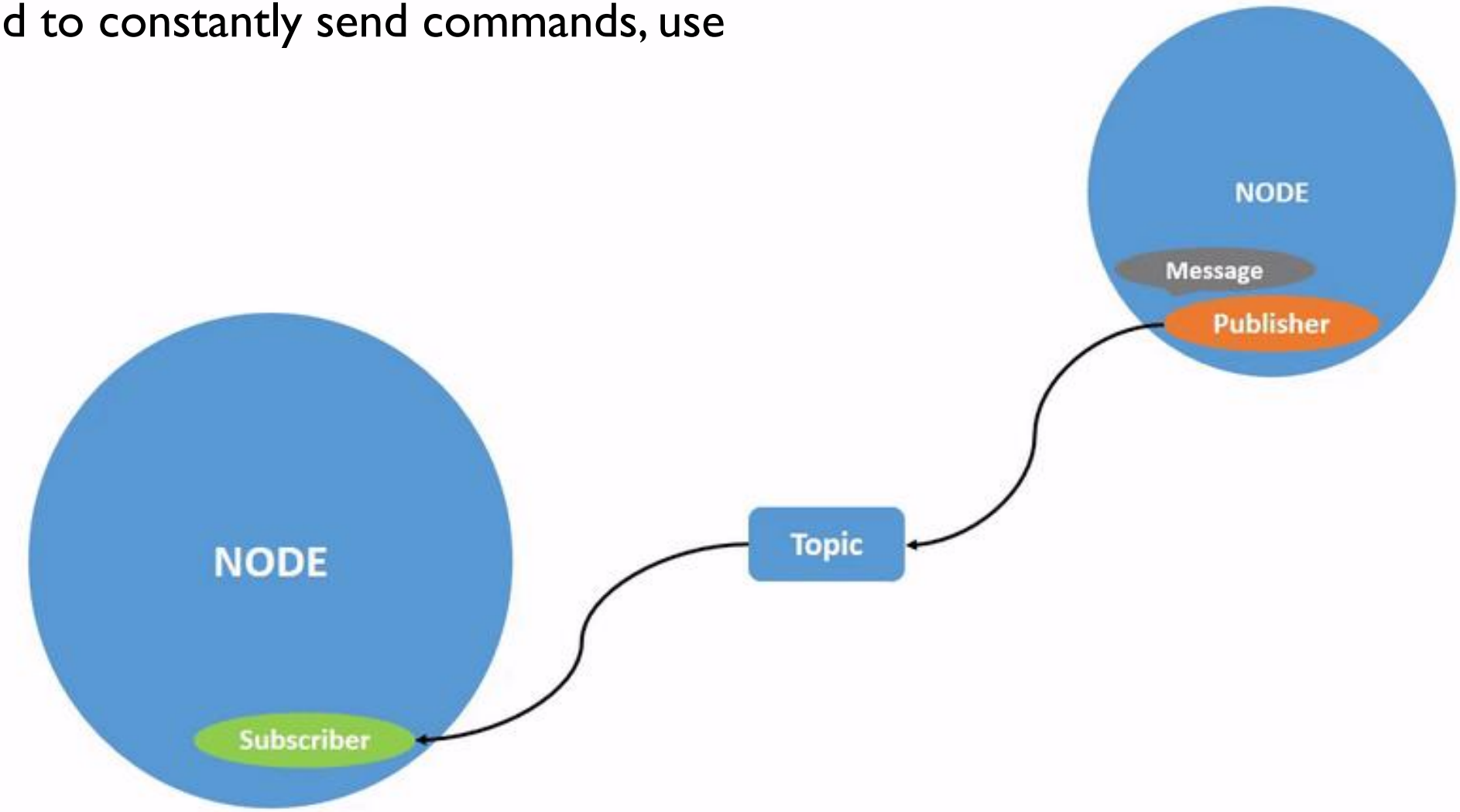
- Services are implemented by two types of nodes: client and server
- A **server** node contains a callback function that provides the service
- A **client** node sends a request which triggers the callback function in the server
- Unlike topics in which messages are sent constantly, services are executed only when a request is made. This is important since there are tasks that take a short time to complete and will waste computation resources if implemented with topics instead of services

Service Examples

- Enable/Disable the motors
- Enable/Disable a sensor (such as Laser Scan, Camera etc)
- Turn a flashlight on/off for environments with varied lighting
- Take a picture with the camera
- Start/Stop video recording with the camera
- Switch on/off LED indicators on the robot
- Get information about a node's current state
- Get on-demand sensor measurement
- Spawn/Delete a model in the simulation
- Open/Close a gripper (end-effector) for material transport

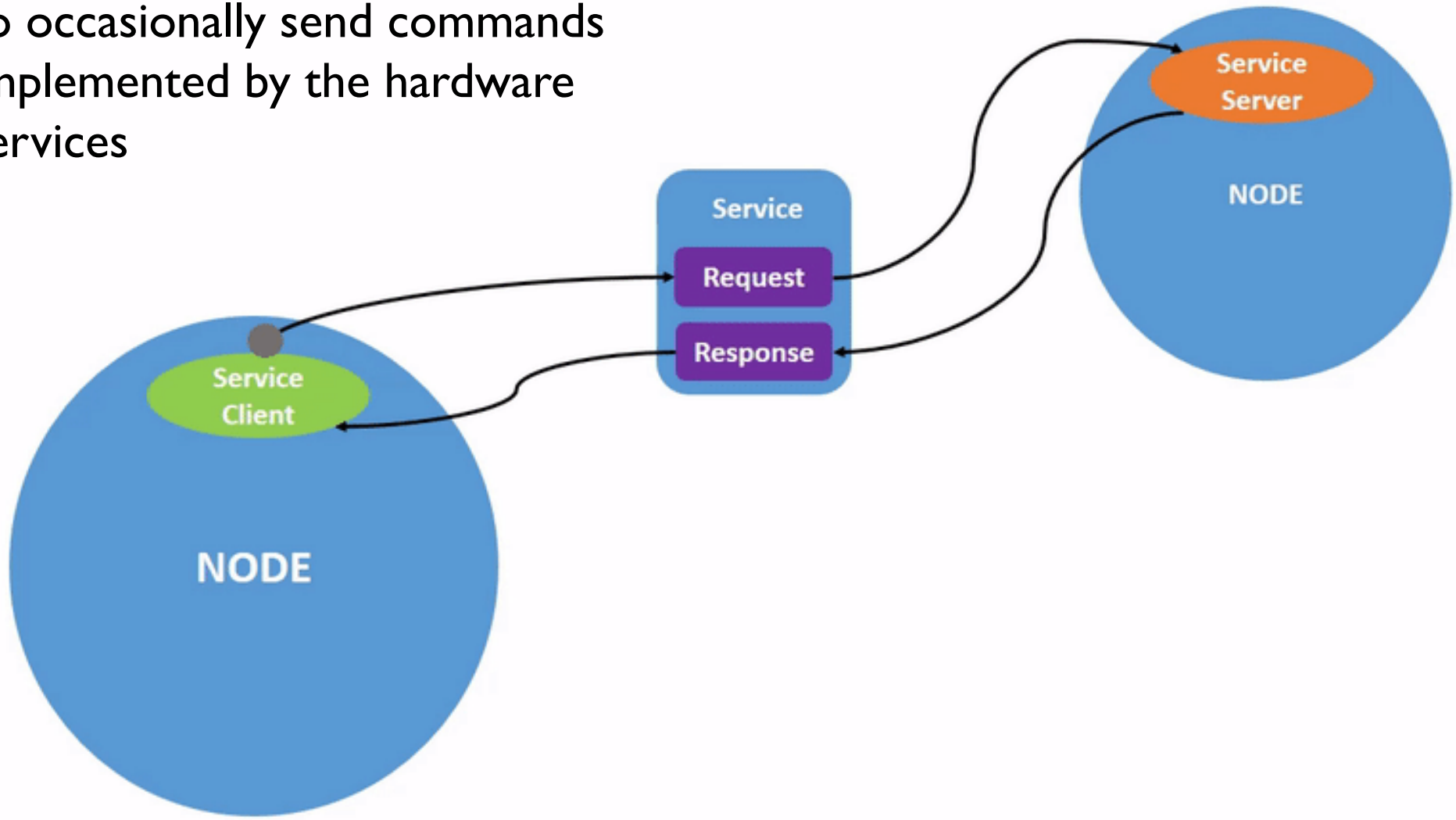
Topic Animation

- If you need to constantly send commands, use topics



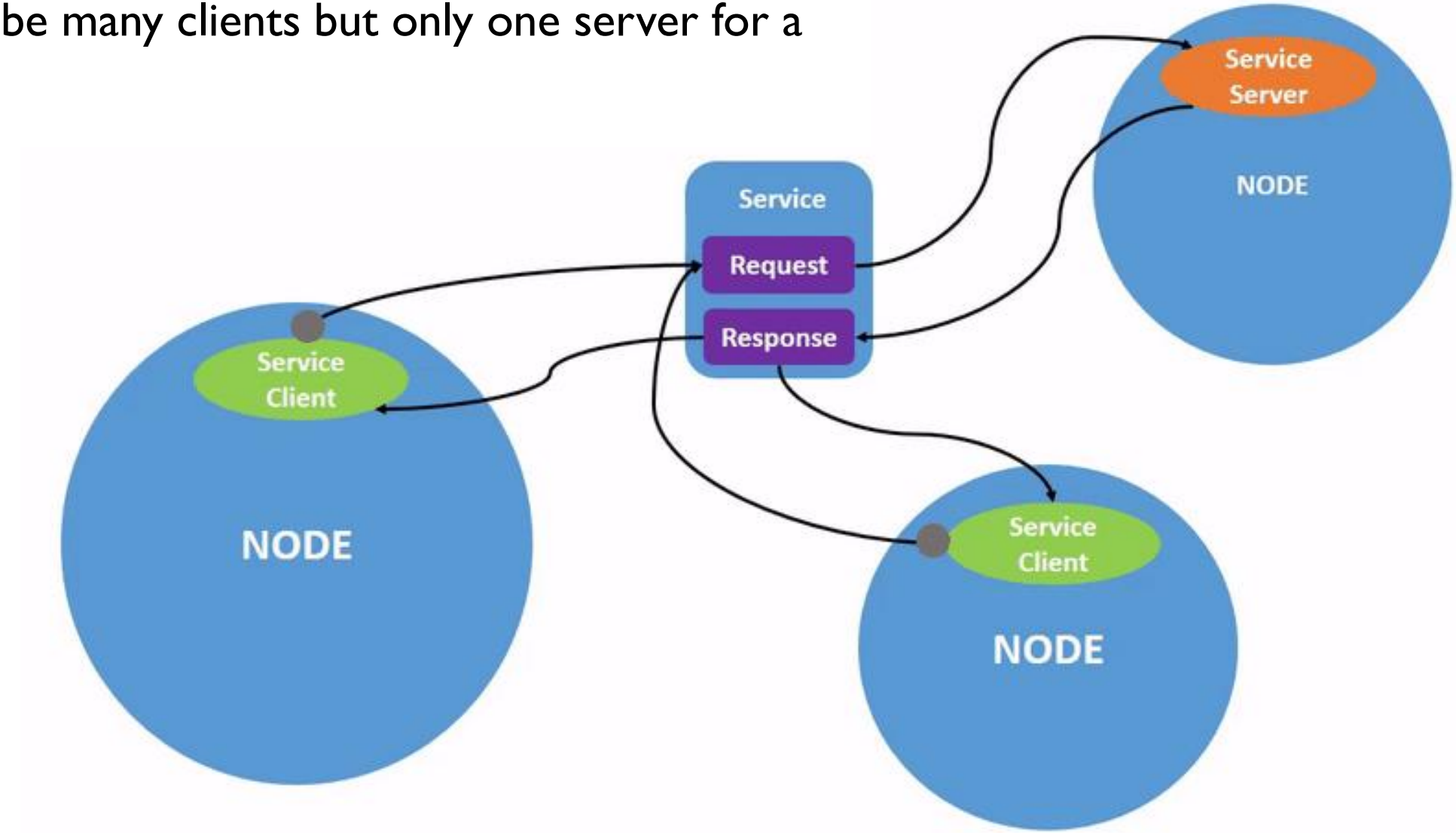
Service Animation

If you need to occasionally send commands that can be implemented by the hardware quickly, use services



Service Animation

There can be many clients but only one server for a service



Service Definition

- A service must be defined in a **.srv** file which is a text file called the service definition file
- A service definition file is used to define the service call inputs (requests) and outputs (response) as shown below:

```
int64 a
int64 b
---
int64 sum
```

Service Definition

- In ROS2, it is currently not possible to make service definitions in a python-based (ament_python) package
- Instead, we will create the definitions in a cpp-based (ament_cmake) package
- Service definitions are placed in an "srv" folder in the cpp-based package
- The cpp_package contains two files: *package.xml* and *CMakeLists.txt* that need to be adjusted for the service definition
- After creating the definition file, the server and client nodes will be created in a python-based package (the cpp-based package is only needed for the service definition)

Communication Summary

- If you need to constantly send commands, use topics
- If you need to occasionally send commands that can be implemented by the hardware quickly, use services
- If you need to occasionally send commands that can take a long or highly variable time, use actions

Lab Tasks

- Download the manual from LMS
- Perform the Lab Tasks as given in the manual and submit it on LMS
- Remember to execute scripts with the terminal