Middlesex University Dubai

PDE4430

ROS – Messages and Services

Middlesex University Dubai

ROS Messages

- So far, we've dealt with basic message type – String
- Information in a ROS application is typically –
  - Floats
  - Integers
  - Strings

- Example: Joint angles with joint names, humidity/temperature information, etc.

ROS Messages

- Along with the basic types, we've also seen Twist – A derived message type

- These are composed of a combination of basic message types:
- **`Twist: Vector3 linear, Vector3 angular`**
- **`Vector3: float64 x, float64 y, float64 z`**

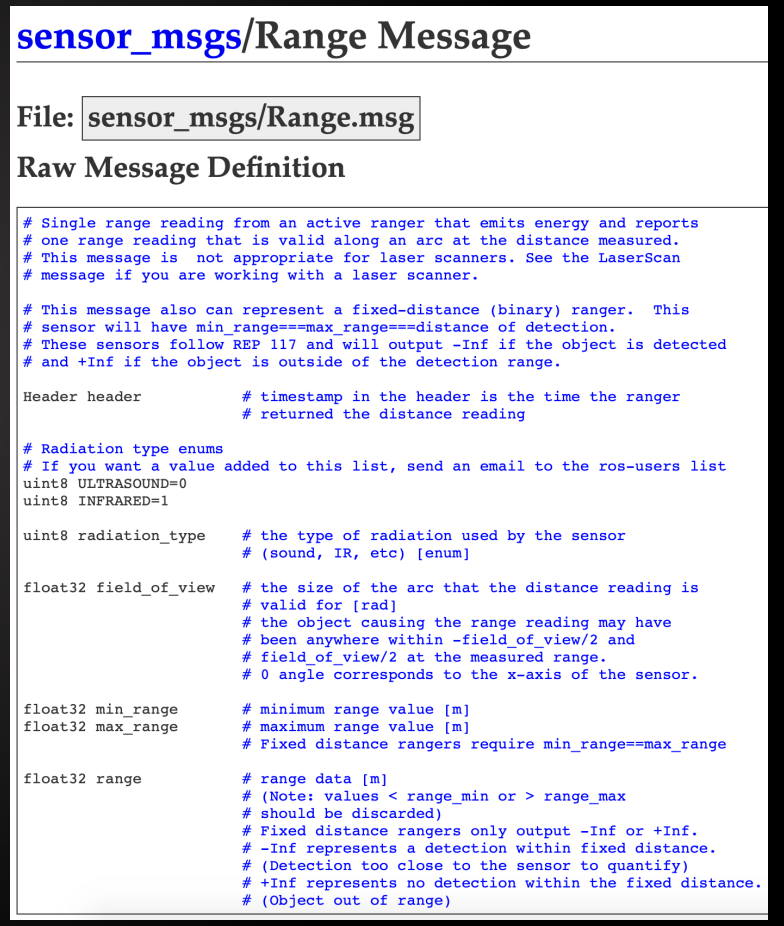
ROS Messages

- Documentation available on the website:

http://wiki.ros.org/common_msgs

http://wiki.ros.org/sensor_msgs

http://wiki.ros.org/std_msgs



sensor_msgs/Range Message

File: sensor_msgs/Range.msg

Raw Message Definition

```
# Single range reading from an active ranger that emits energy and reports
# one range reading that is valid along an arc at the distance measured.
# This message is  not appropriate for laser scanners. See the LaserScan
# message if you are working with a laser scanner.

# This message also can represent a fixed-distance (binary) ranger.  This
# sensor will have min_range===max_range===distance of detection.
# These sensors follow REP 117 and will output -Inf if the object is detected
# and +Inf if the object is outside of the detection range.

Header header           # timestamp in the header is the time the ranger
                        # returned the distance reading

# Radiation type enums
# If you want a value added to this list, send an email to the ros-users list
uint8 ULTRASOUND=0
uint8 INFRARED=1

uint8 radiation_type    # the type of radiation used by the sensor
                        # (sound, IR, etc) [enum]

float32 field_of_view   # the size of the arc that the distance reading is
                        # valid for [rad]
                        # the object causing the range reading may have
                        # been anywhere within -field_of_view/2 and
                        # field_of_view/2 at the measured range.
                        # 0 angle corresponds to the x-axis of the sensor.

float32 min_range       # minimum range value [m]
float32 max_range       # maximum range value [m]
                        # Fixed distance rangers require min_range==max_range

float32 range           # range data [m]
                        # (Note: values < range_min or > range_max
                        # should be discarded)
                        # Fixed distance rangers only output -Inf or +Inf.
                        # -Inf represents a detection within fixed distance.
                        # (Detection too close to the sensor to quantify)
                        # +Inf represents no detection within the fixed distance.
                        # (Object out of range)
```

ROS Messages

- Important to know the structure of a ROS message
- Typical notation – Defined by TWO things:
  - Package name it belongs to
  - It's own name

`PackageName/MessageName`

- Example: **`std_msgs/String`**
- Example: **`geometry_msgs/Twist`**

Middlesex University Dubai

ROS Messages

- Each ROS message also has data – A type and a field name
- Examples:
  - **`std_msgs/String –`**
    - **`string data`**
  - **`Geometry_msgs/Vector3 –`**
    - **`float64 x`**
    - **`float64 y`**
    - **`float64 z`**

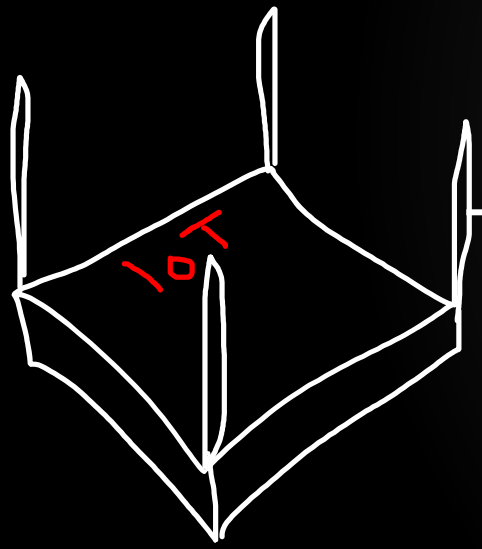
ROS Messages

- Thus, for a custom-defined ROS message, you need:
  - A package (name)
  - A message type
  - Constituent data types and fields

- Let’s create our own message type now

ROS Messages – Example

- We have a new IoT sensor that detects temperature and humidity.



PUBLISHER

/topic

IoTSensor Message

- ID
- Name
- Temperature
- Humidity

SUBSCRIBER

ROS Messages – Example

- Nothing like this data type is defined in ROS

- We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

Middlesex University Dubai

ROS Messages – Example

- Nothing like this data type is defined in ROS

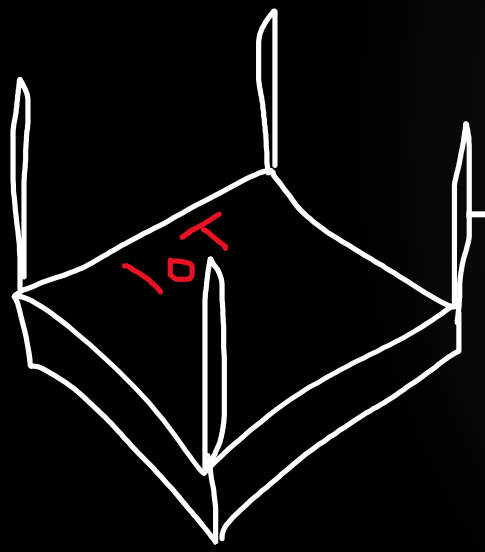
- We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

Middlesex University Dubai

ROS Messages – Overview of Steps

1. Create **`msg`** folder in your package
2. Create the custom message file **`(.msg)`** in the folder
3. Define the elements of the message type in the file
4. Update dependencies
   - **`CMakeLists.txt`**
   - **`Package.xml`**
5. Compile package using **`catkin_make`**
6. Verify using **`rosmsg show`**

ROS Messages – Example



PUBLISHER

/topic

PUBLISHER

IoTSensor Message

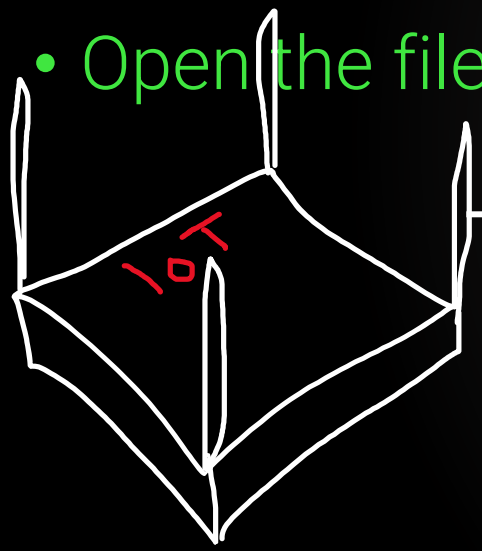
- ID: `int`
- Name: `string`
- Temperature: `float32`
- Humidity: `float32`

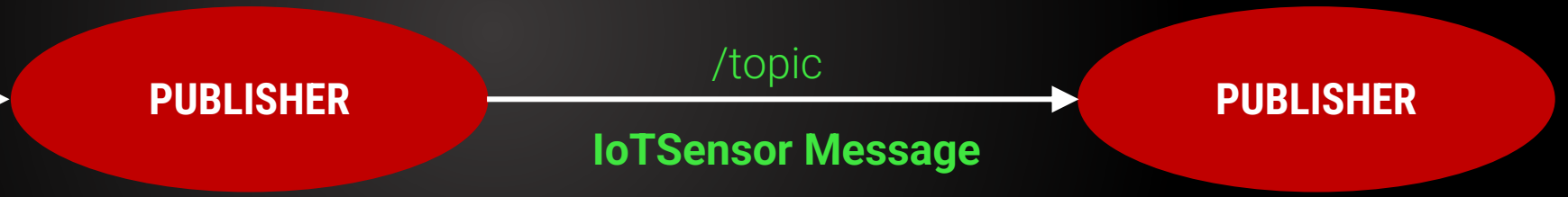
ROS Messages – Step 1

- Create `msg` folder in the package root directory
- Use Terminal or GUI – Up to you

ROS Messages – Steps 2 & 3

- Create file called **`IoTSensor.msg`** within the folder
- Open the file in VS Code





IoTSensor Message

- ID: **`int`**
- Name: **`string`**
- Temperature: **`float32`**
- Humidity: **`float32`**

ROS Messages – Steps 2 & 3

- Write the following lines:

```
int32 id
string name
float32 temperature
float32 humidity
```

- Go to https://wiki.ros.org/msg for more data types

ROS Messages – Step 4

- Update dependencies – Very important
- Won't work if this isn't done properly
- Other packages can use this type as well – So it needs to be defined correctly
- Need to update dependencies in two places –
  - **`CMakeLists.txt`**
  - **`Package.xml`**

- Let's update Package.xml first

ROS Messages – Step 4a – Package.xml

- Open **`Package.xml`** in VS Code
- Make sure you have the following two lines mentioned:

```
<build_depend>message_generation</build_depend>
<execution_depend>message_runtime</execution_depend>
```

ROS Messages – Step 4b – CMakeLists.txt

- More steps in this file. Pay close attention.

- Open **`CMakeLists.txt`** in VS Code
- Add **`message_generation`** to the **`find_package`** list
- Uncomment **`add_message_files`** and write **`IoTSensor.msg`** in the list
- In **`catkin_package`** add **`message_runtime`** to **`CATKIN_DEPENDS`**
- Uncomment **`generate_messages`**

ROS Messages – Step 5 & 6

- Build the package using **`catkin_make`**

- Test if the message was built correctly by using:

```
rosmsg show IoTSensor
```

Middlesex University Dubai

ROS Messages – Example

- Nothing like this data type is defined in ROS

- We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Publisher

- We will modify the code from **`talker.py`** to make the IoT Sensor publisher
- Copy the code
- Paste it in a new file called **`iot_sensor_publisher.py`**
- Open this new file in VS Code so we can start modifying it

ROS Messages – Publisher

- Import the message type:

```
from pde4420_session2.msg import IoTSensor
```

- Create Publisher – Change name and topic type:

```
pub = rospy.Publisher('iot_sensor_topic',
        IOTSensor, queue_size = 10)
```

- Change node name

ROS Messages – Publisher

- Inside the while loop, initialize an object of type IoTSensor:

```
iot_sensor = IoTSensor()
iot_sensor.id = 1
iot_sensor.name = “iot_01”
iot_sensor.temperature = 35.28 + (random.random() *
5)
iot_sensor.humidity = 67.5 + (random.random() * 3)
rospy.loginfo(“I publish: “)
rospy.loginfo(iot_sensor)
pub.publish(iot_sensor)
```

ROS Messages – Publisher

- Make the python file executable
- Another way to do it:
  - Right-click the .py file
  - Properties
  - Permissions Tab
  - “Make the file executable”
- If you run the command **`ls -l`** and you see the file name in green, it’s executable. (**`-l`** stands for long-listing format – More details visible)

- Run the file and examine the nodes/topics

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
- We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
✓We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
✓We will create a publisher that can publish this message type
- We will create a subscriber that can listen to this message type

ROS Messages – Subscriber

- Similar procedure
- Open **`listener.py`** and copy/paste the code into a new file
- Save it as **`iot_sensor_subscriber.py`**

- Changes are also similar

ROS Messages – Subscriber

- Replace **`String`** with **`IoTSensor`**
- Modify the callback –
  - **`def iot_sensor_callback(iot_sensor_message):`**

```
rospy.loginfo("IoT Data: (%d, %s, %.2f, %.2f)",
iot_sensor_message.id, iot_sensor_message.name,
iot_sensor_message.temperature,
iot_sensor_message.humidity)
```

ROS Messages – Subscriber

- Make the file executable
- Run the file
- Examine the nodes/topics

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information

✓We will create a publisher that can publish this message type

- We will create a subscriber that can listen to this message type

ROS Messages – Example

- Nothing like this data type is defined in ROS

✓We will create our own message type to publish this information
✓We will create a publisher that can publish this message type
✓We will create a subscriber that can listen to this message type

Break

Middlesex University Dubai

ROS Services

ROS Topics – Need for Something Else

- We've seen Publishers and Subscribers communicate via topics
- Topics are many-to-many **ONE–WAY** communication infrastructures
- Similar to 100s of email newsletters you receive ***constantly*** (that we "subscribe" to)
- But do we read all the newsletters? No. We "process" only a few
- Topics are undoubtedly great, BUT there are a few limitations – Not to undermine topics, but just to point out the limitations due to the way they are built

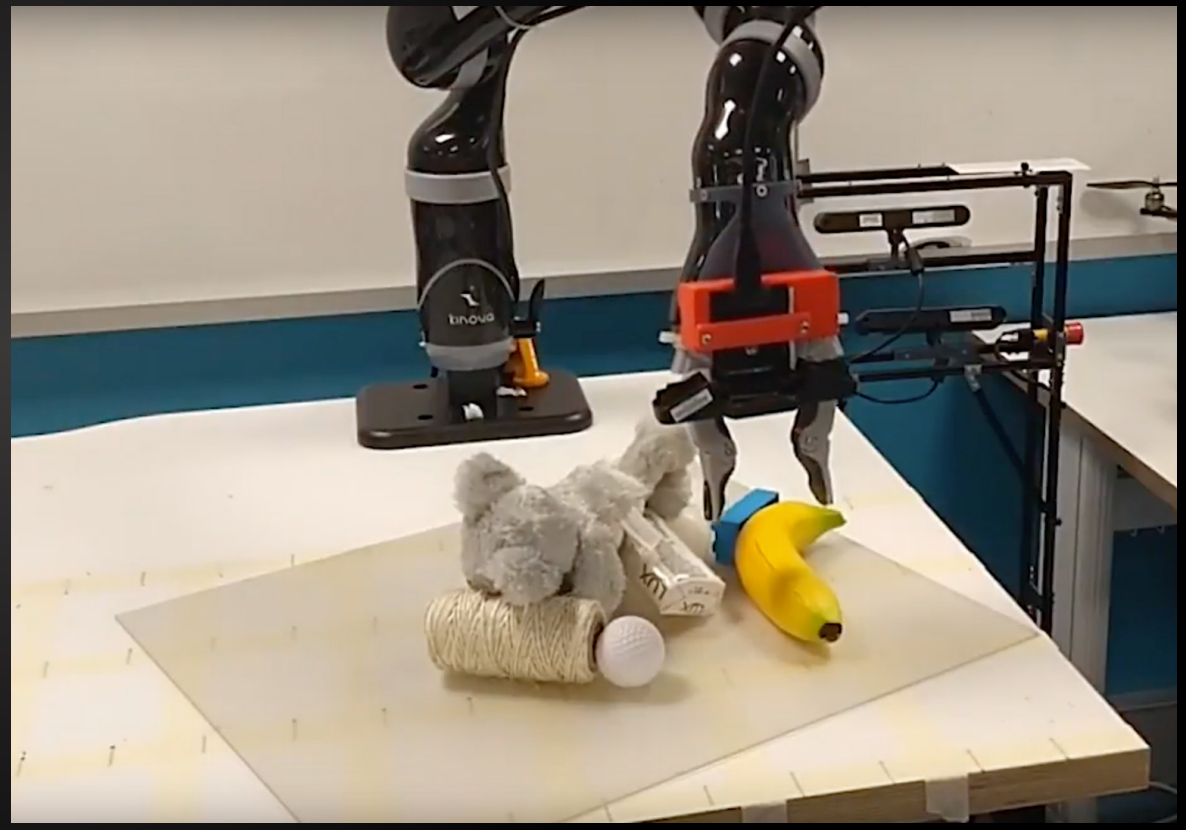
ROS Topics – Need for Something Else

- No app–level acknowledgement when exchanging the actual information via topics
- Works really well when a sensor is constantly publishing information, or monitoring a certain situation
- But it may not be efficient in all situations – 2 main scenarios
  - When it is important to receive acknowledgement ***when*** a certain message was received
  - When the topic may end up consuming too much data bandwidth

ROS Topics – Need for Something Else

- Example: If a robotic arm is manipulating objects in a static scene, no point to constantly publish camera data – Even at a low frequency
- It's like watching a movie with only one still frame
- In this case, it's more efficient to get a snapshot of the scene before you start manipulating things

- THIS is where ROS services come in!



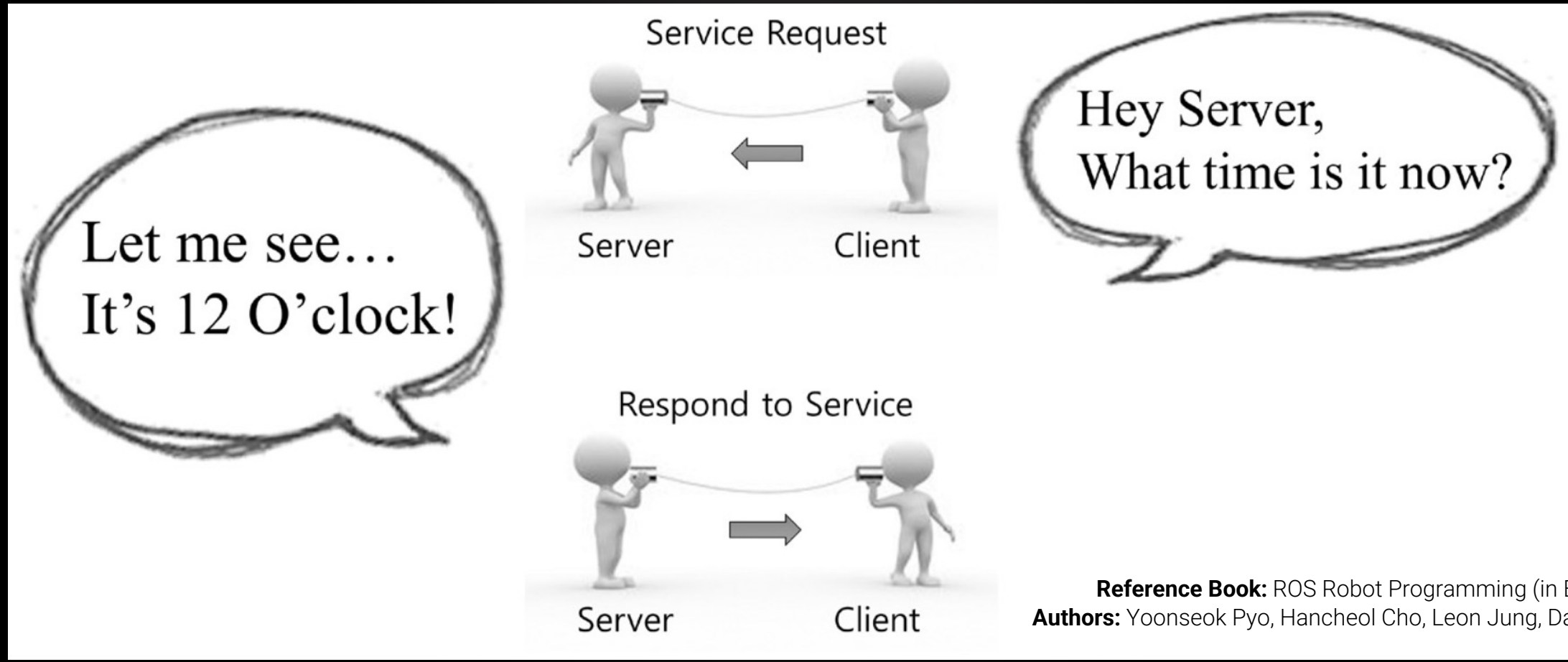
Middlesex University Dubai

ROS Services

- They provide a client/server architecture
- ROS services work via request–response communication
- It is NOT a continuous communication – It is only ONE time
- After the response communication is closed
- In a broader sense, ROS services are a gateway to “event based” execution

- A ROS service is defined using 2 message types:
  - A Request Message type
  - A Response Message type

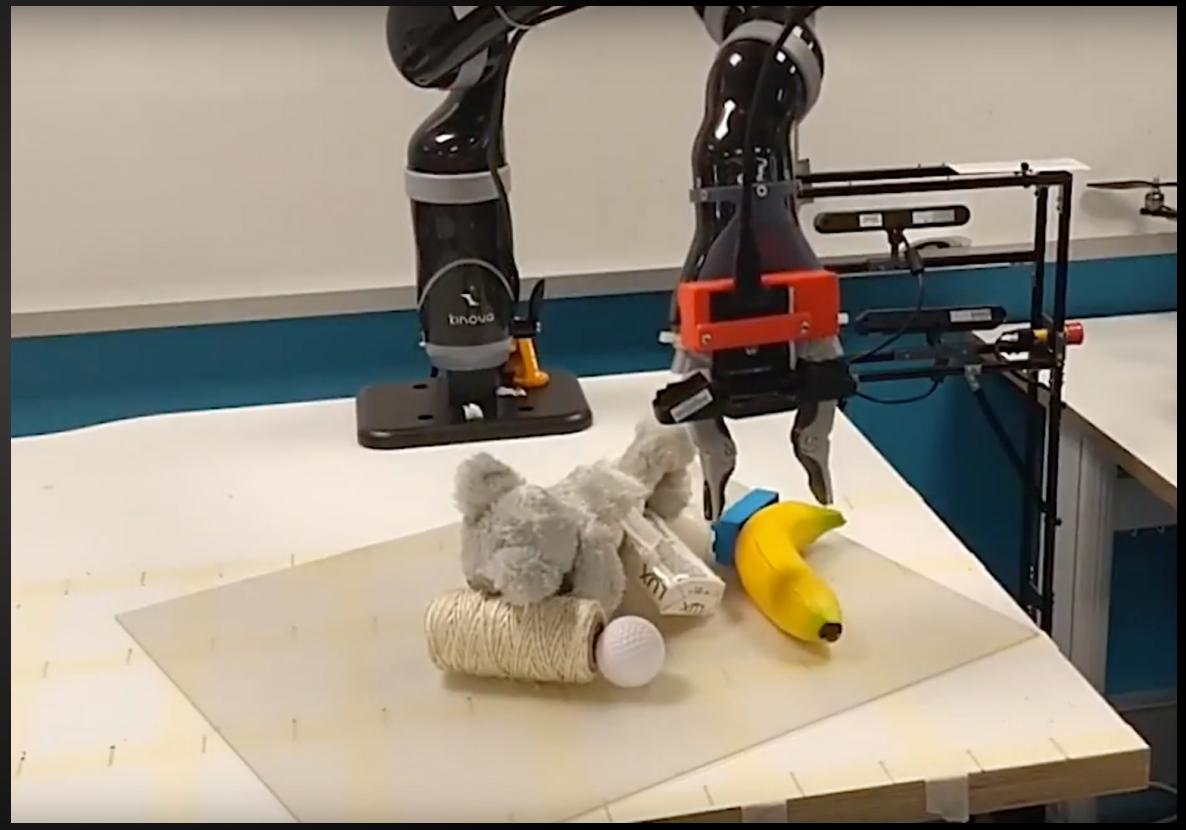
ROS Services - Illustration



Reference Book: ROS Robot Programming (in E
Authors: Yoonseok Pyo, Hancheol Cho, Leon Jung, Da

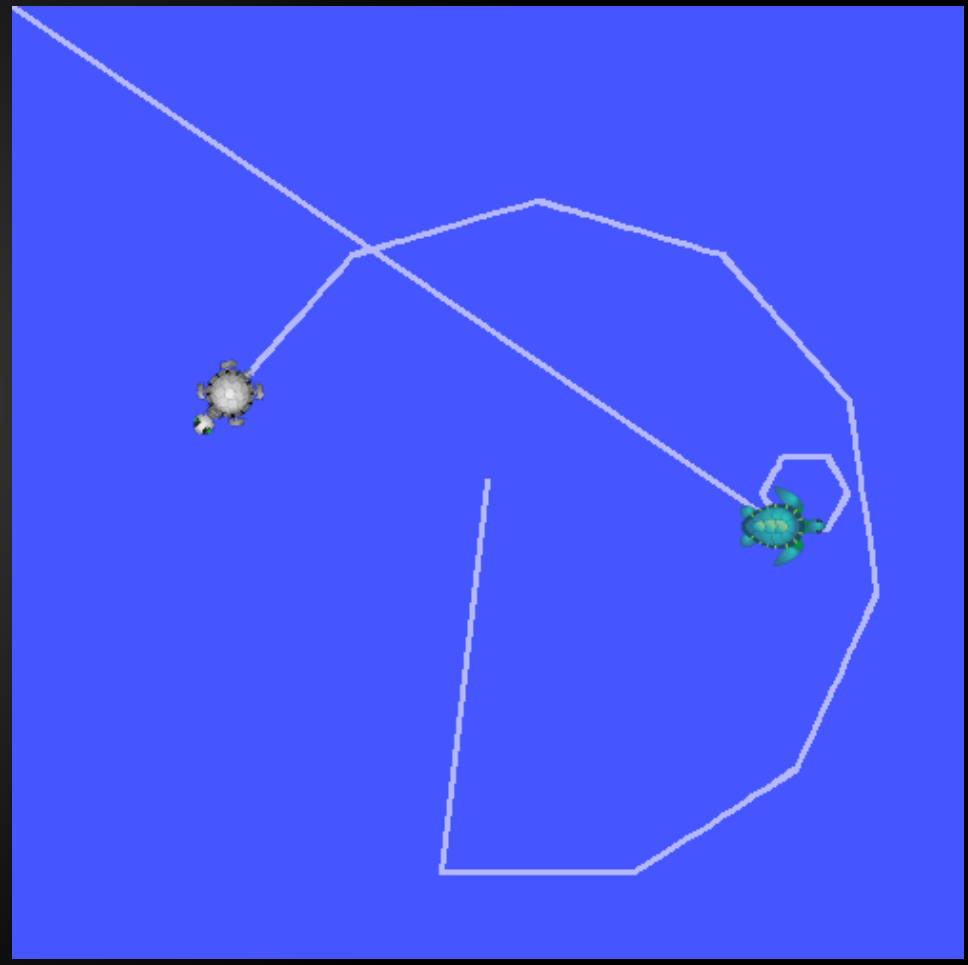
ROS Services - Example

- In the manipulator example – A request message type would consist of a command to the camera to trigger a snapshot of the scene
- Response message type would consist of one image of the scene



ROS Services – Other Examples

- Other examples?
- Path Planning – Find me a path from point A to B
- Spawn another robot – Create new turtle robot in Turtlesim



ROS Services – Hands on

- Let's fire up that Turtle again **`(roscore, rosrun)`**
- To see a list of ROS services:

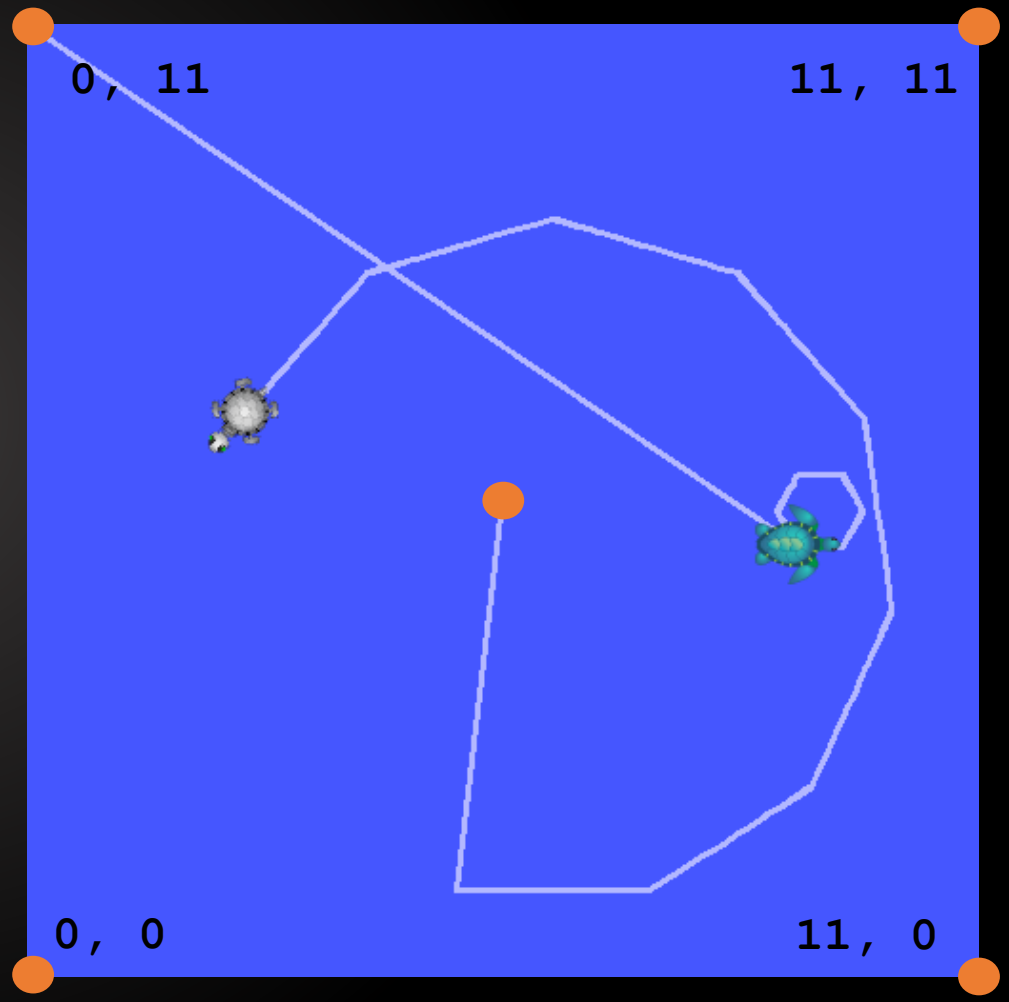
```
rosservice list
```

- To see more information about a specific service (Type & Args):

```
rosservice info /spawn
```

ROS Services – Hands on

- The blue Turtlesim window coordinate system is represented as [x, y]
- Goes from [0, 0] to [11, 11]



ROS Services – Hands on

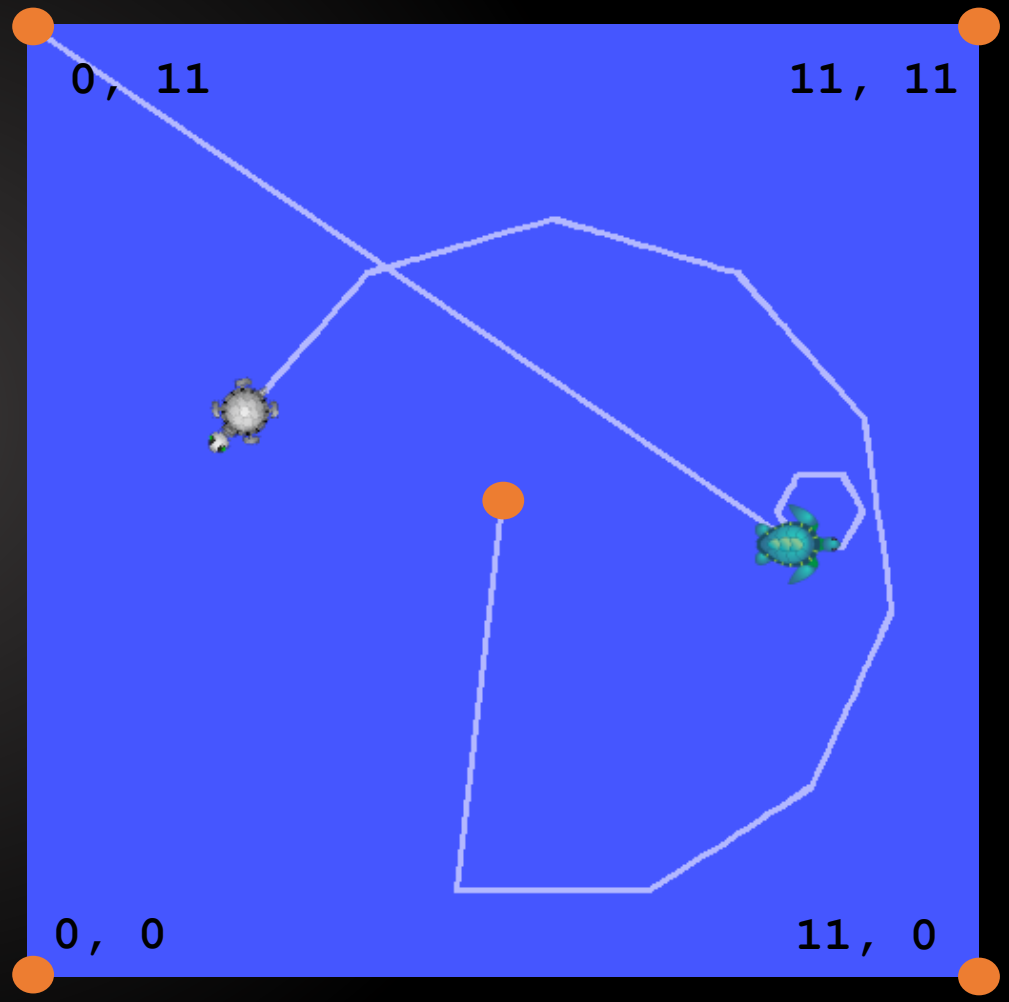
- To see more information about the data structure:

```
rossrv info
turtlesim/Spawn
```

```
float32 x
float32 y
float32 theta
string name
---
string name
```

Client (Request) needs to provide this information

Server (Response) sends this back



ROS Services - Hands on

- Call a service using the following format:

```
rosservice call /serviceName arg1 arg2 arg3
```

- Spawn a new Turtle by typing the following command:

```
rosservice call /spawn 2 6 90 t2
```

- Spawn yet another one by calling:

```
rosservice call /spawn 9 2 180 t3
```

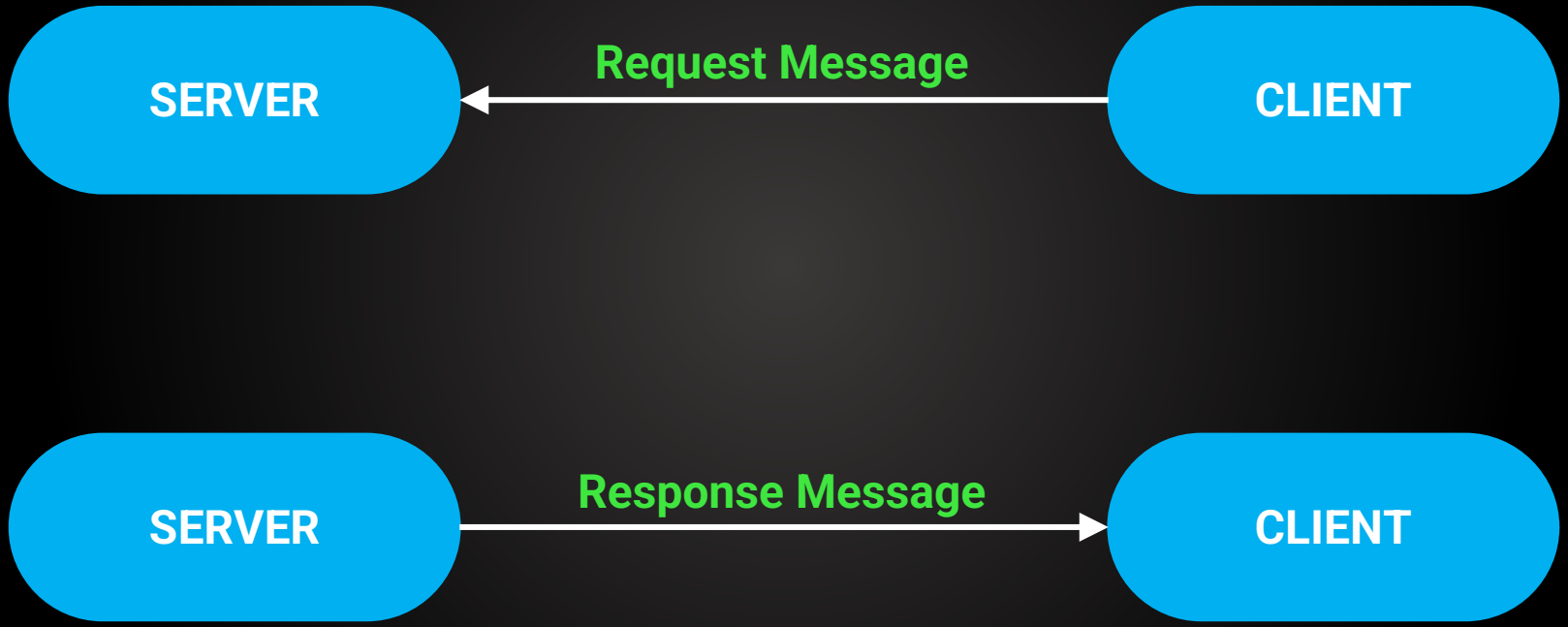
ROS Services

- What do the other services do?
- Try finding out about the services **`/reset`** and **`/kill`**
- Call them one by one and see what happens

Middlesex University Dubai

Steps to Write a ROS Service

ROS Service – Sequence



ROS Service – Sequence

- The process starts when the server wakes up and starts listening for requests
- Then the client sends a service request to the server
- The server parses the request and processes it
- Finally, the server sends a response to the client

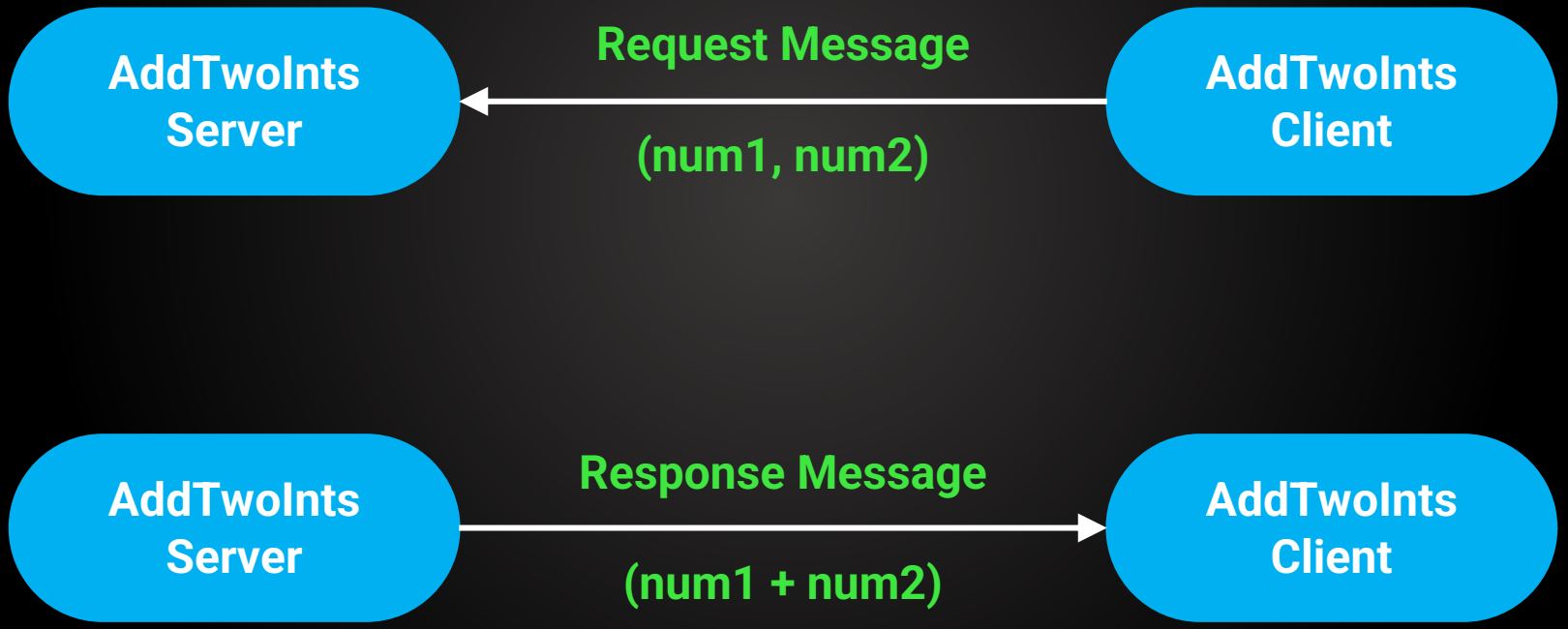
- As mentioned earlier, the difference between a service and Publisher/Subscriber is that here we define TWO message types instead of one (the topic)

ROS Service – Steps

1. Define the Service message (`.srv` file)
   - Define the request, the response, and the type of data they'll carry out
2. Create the ROS server node
3. Create the ROS client node
4. Execute the service
5. Consume the result of that service by the client

ROS Service – Example

- Let's create a service to add two integers



ROS Service – Example

- For this example we need to define the service request and response:

- Request: 2 arguments – num1 and num2
- Response: 1 argument – sum

ROS Service – Example – Step 1

Create `.srv` file

- Open your package and create a folder called **`srv`**
- Create a file called **`AddTwoInts.srv`**
- Open this file in VS Code

```
int64 num1
int64 num2
---
int64 sum
```

ROS Service – Example – Step 1

- Make sure our dependencies are correct
- Same as message dependencies, in **`Package.xml`** and **`CMakeLists.txt`**
- In addition, in **`CMakeLists.txt`**, uncomment the line:

```
add_service_files (FILES…
```

- Write **`AddTwoInts.srv`** in the list
- Build the package

ROS Service – Example – Step 1

- To verify that the service was created successfully, go to **`workspace/devel/include/packageName`**

- You should see three files there:
  - **`AddTwoInts.h`**
  - **`AddTwoIntsRequest.h`**
  - **`AddTwoIntsResponse.h`**

- Another way to verify: **`rossrv list`** OR **`rossrv show AddTwoInts`**

ROS Service – Example – Step 2

Create the server

- This is similar to the subscriber in the sense that it “listens”, but the difference is that it will send a response as well
- Similar also because the server has a callback function that processes the request

ROS Service – Example – Step 2

```
#!/usr/bin/env python
from packageName.srv import AddTwoInts
from packageName.srv import AddTwoIntsRequest
from packageName.srv import AddTwoIntsResponse
```

- We need to import all three

ROS Service – Example – Step 2

```
import rospy

def handle_add_two_ints(req):
    print "Returning [%s + %s = %s]"%(req.num1,
    req.num2, (req.num1 + req.num2))
    return AddTwoIntsResponse(req.num1 + req.num2)
```

- Create the callback function
- **`req`** has two fields – num1 and num2
- Sends the response in the last line

ROS Service – Example – Step 2

```
def add_two_ints_server():
      rospy.init_node('add_two_ints_server')
      s = rospy.Service('add_two_ints', AddTwoInts,
      handle_add_two_ints)
      print("Ready to add two ints")
      rospy.spin()

if __name__ == "__main__":
      add_two_ints_server()
```

- Create the initialization function

ROS Service – Example – Step 2

```
def add_two_ints_server():
        rospy.init_node('add_two_ints_server')
        s = rospy.Service('add_two_ints', AddTwoInts,  handle_add_two_ints)
        print("Ready to add two ints")
        rospy.spin()

if __name__ == "__main__":
        add_two_ints_server()
```

- Create the initialization function
- **'add_two_ints'** is the service name
- **AddTwoInts** is the type
- **handle_add_two_ints** is the callback function

ROS Service – Example – Step 3

Create the client

```
#!usr/bin/env python

import sys
import rospy
from packageName.srv import AddTwoInts
from packageName.srv import AddTwoIntsRequest
from packageName.srv import AddTwoIntsResponse
```

ROS Service – Example – Step 3

```
def add_two_ints_client(x, y):
        rospy.wait_for_service('add_two_ints')
        try:
                add_two_ints = rospy.ServiceProxy('add_two_ints',
        AddTwoInts)
                resp = add_two_ints(x, y)
                return resp.sum
        except rospy.ServiceException, e:
                print "Service call failed: %s"%e
```

- Wait for service first – We defined the service name in the server
- Create a client
- Call it and save the result
- Return the sum to the main function

ROS Service – Example – Step 3

```
def usage():
        return "%s [x y]"%sys.argv[0]
if __name__ == "__main__":
        if len(sys.argv) == 3:
                x = int(sys.argv[1])
                y = int(sys.argv[2])
        else:
                print usage()
                sys.exit(1)
        print "Requesting %s + %s"%(x, y)
        print "%s + %s = %s"%(x, y, add_two_ints_client(x, y))
```

- Get numbers from the terminal
- If no numbers received, throw an error and quit
- Otherwise call the function that calls the service

ROS Service – Example – Step 4

Run Everything

- `roscore`
- `rosrun packageName add_server.py`
- `rosrun packageName add_client.py` 

- `rosrun packageName add_client.py 5 12`

Middlesex University Dubai

Exercise

ROS Service Exercise

Develop a ROS service that provides the area of a rectangle, when a client sends the length and width.

Notes:

- Create a new package for this exercise (This means you have to fix the dependencies) – **`ros_service_assignment`**
- Use the service file: **`RectangleArea.srv`**
- Create server in **`rect_server.py`**; Client in **`rect_client.py`**
- Use **`float32`** for width and height
- Again, pay close attention to the dependencies. Please.

Middlesex University Dubai

Solution

Middlesex University Dubai

Solution – Server

```python
#!/usr/bin/env python

from ros_service_assignment.srv import RectangleArea
from ros_service_assignment.srv import RectangleAreaRequest
from ros_service_assignment.srv import RectangleAreaResponse

import rospy

def rectangle_area_callback(req):
    print "Returning area of a rectangle [%s * %s = %s]"%(req.width, req.height, (req.width * req.height))
    return RectangleAreaResponse(req.width * req.height)

def rectangle_area_server():
    rospy.init_node('rectangle_area_server_node')
    s = rospy.Service('rectangle_area_service', RectangleArea, rectangle_area_callback)
    print "Ready to calculate the area of a rectangle."
    rospy.spin()

if __name__ == "__main__":
    rectangle_area_server()
```

Solution – Client

```python
#!usr/bin/env python

import sys
import rospy
from ros_service_assignment.srv import RectangleArea
from ros_service_assignment.srv import RectangleAreaRequest
from ros_service_assignment.srv import RectangleAreaResponse

def request_rectangle_area(x, y):
    rospy.wait_for_service('rectangle_area_service')
    try:
        calculate_area = rospy.ServiceProxy('rectangle_area_service', RectangleArea)
        server_response = calculate_area(x, y)
        return server_response.area
    except rospy.ServiceException, e:
        print "Service call failed %s" % e

def usage():
    return "%s [x y]"%sys.argv[0]

if __name__ == "__main__":
    if len(sys.argv) == 3:
        x = float(sys.argv[1])
        y = float(sys.argv[2])
    else:
        print usage()
        sys.exit(1)
    print "Requesting area of rectangle with width = %s and height = %s"%(x, y)
    print "Area of the rectangle (%s x %s) is: %s"%(x, y, request_rectangle_area(x, y))
```