Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen
International Center for
Information Technology

ROS Services

Foundation Course

March 17, 2020

Hassan Umari

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

Recap

Summary of yesterday's session

1. We created a ROS **package** and wrote a ROS **node** in Python.
2. We saw how to define a **publisher** and a **subscriber** in our node.
3. We looked into **launch** files, and ROS **parameters**.
4. We saw how **node names** (or any resource: parameter, topic ..etc) are resolved.
5. and how to do name **remapping**.

Recap

What does this node do?

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

import rospy
from std_msgs.msg import String

def myfunction(received_msg):
    msg.data = "Hey, I think I heard: " + received_msg.data
    pub.publish(msg)

if __name__ == ’__main__’:
    rospy.init_node(’useless_node’)

    pub = rospy.Publisher(’mouth’, String, queue_size=1)
    sub = rospy.Subscriber(’ears’, String, callback=myfunction)

    msg = String()
    rospy.spin()
```

Recap

Summary of yesterday's session

We notice the following:

1. The previous node communicates back and forth with another subscribing node.
2. But is there a better way to make two-way communication between nodes?

Recap

Summary of yesterday's session

Yes, we can do that with:

1. ROS services.
2. ROS actions.

Recap

ROS Concepts

Concepts related to ROS computation graph:

1. ~~Nodes.~~ ✓
2. ~~Topics.~~ ✓
3. ~~Messages.~~ ✓
4. ~~Master.~~ ✓
5. Services.
6. Actions
7. ~~Parameter Server.~~ ✓
8. Bags.

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

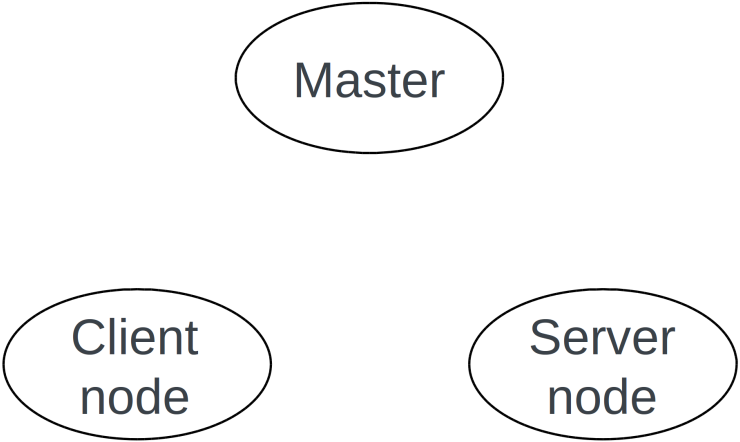
2.5 Using a Custom Service

3. References

ROS Services

Services:

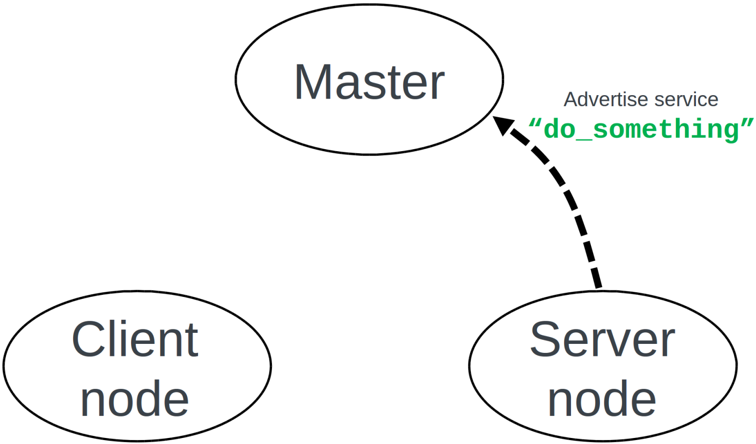
- Communication happens between two nodes, the service **server** node, and the service **client** node.
- A Client node sends a request for a named service and waits for the response, a node serving this service responds, and the communication is over.
- it is a one-to-one, two-way, one-time communication.

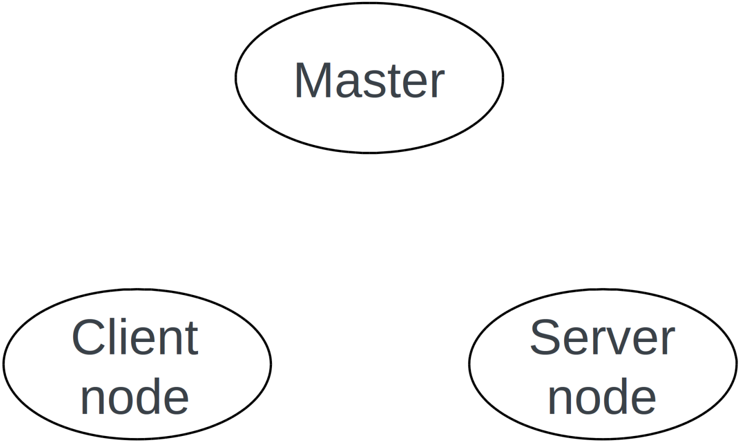


Master

Client node

Server node

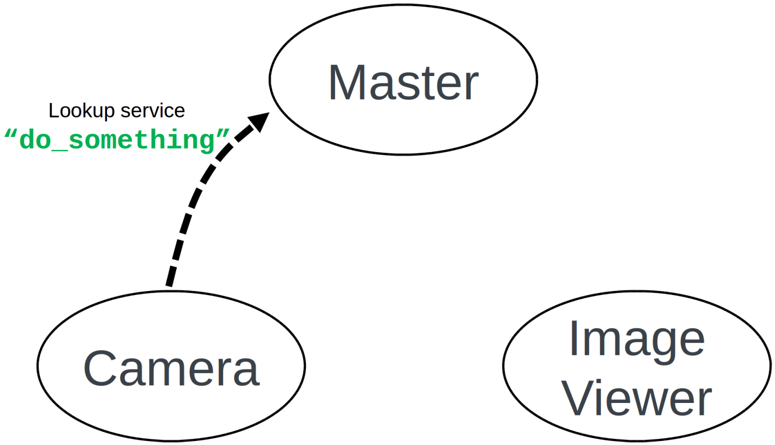

Master
Advertise service
"do_something"
Client node
Server node

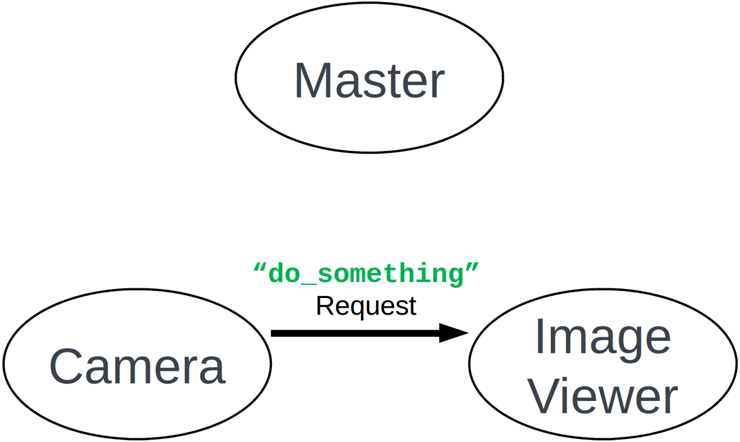


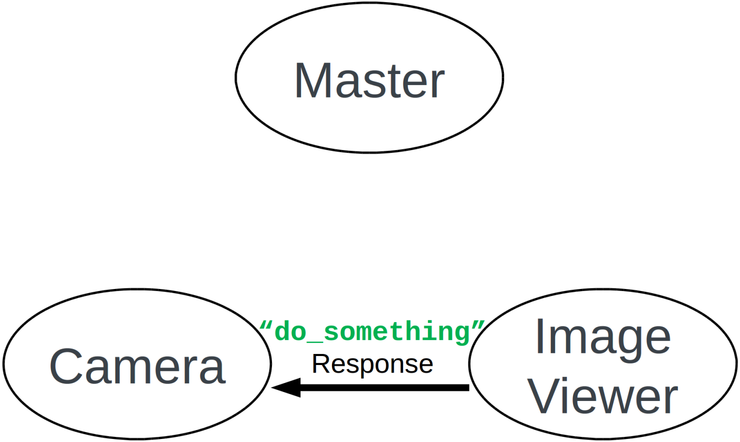
Master

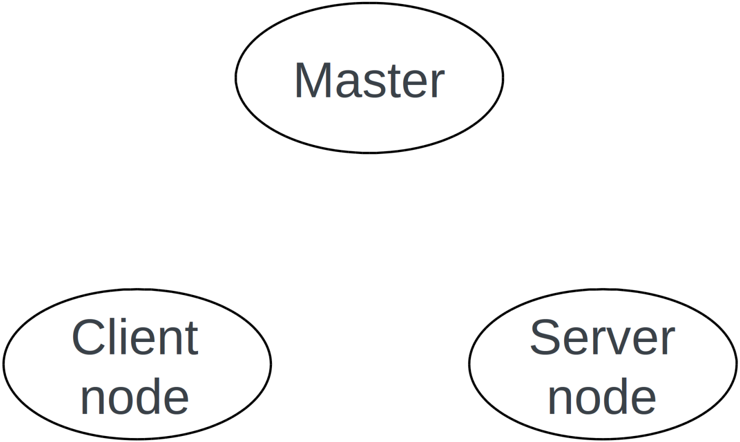
Client node

Server node


Master
Lookup service
“do_something”
Camera
Image Viewer


Master
“do_something”
Request
Camera
Image Viewer


Master
Camera
"do_something"
Response
Image Viewer



Master

Client node

Server node

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

ROS Services

TurtleSim Services

- The `turtlesim` node also acts as a server of multiple ROS services.
- To check for ROS services currently being served:

```
rosservice list
```

Example
(TurtleSim services)

ROS Services

TurtleSim Services

- To see more information on a service:

```
rosservice info <service name>
```

- Example:

```
rosservice info /turtle1/set_pen
```

ROS Services

TurtleSim Services

- To see information on a service type:

```
rossrv show <package/srv>
```

- Example:

```
rossrv show turtlesim/SetPen
```

ROS Services

TurtleSim Services

- The output to previous command:

```
uint8 r
uint8 g
uint8 b
uint8 width
uint8 off
---
```

ROS Services

TurtleSim Services

- To call a service:

```
rosservice call <service name> <request>
```

- Example:

```
rosservice call /turtle1/set_pen "r:  255, g:  255, b:  0, width:  5, ’off’:  0"
```

- Note: use command auto-completion by clicking Tab key twice.

Exercise 1

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

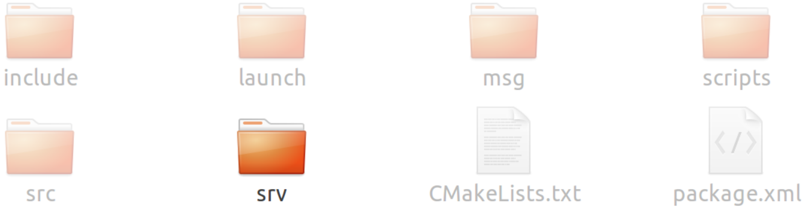
2.5 Using a Custom Service

3. References

Service Description Files

.srv file format

- A service is defined in a text file with `.srv` extension.
- `.srv` files must be placed in `srv` folder inside the package.



Service Description Files

.srv file format

- To see the services defined in a package:

```
rossrv package <package name>
```

- Example

```
rossrv package turtlesim
```

Service Description Files

.srv file format

- The file consists of two parts: **request** message, and **response** message.
- request and response are separated with "---".

```
fieldtype    fieldname
fieldtype    fieldname
---
fieldtype    fieldname
fieldtype    fieldname
```

- Field type can be a ROS built-in type or a defined ROS message. Built-in types

Service Description Files

.srv file format

- We saw earlier the `turtlesim/SetPen` service description:

```
uint8    r
uint8    g
uint8    b
uint8    width
uint8    off
---
```

Service Description Files

.srv file format

- Question: if we modify our "useless" node to use services...

  how would the service file look like?

Service Description Files

.srv file format

- It could be something like this:

```
string    str
---
string    str
```

Service Description Files

.srv file format

- When you build your package, Catkin reads `.srv` files and generates Python classes for you.
- you can use the generated Python classes to define a service server, or service client in your node.
- We will write a custom service file and do the build process later (but it's exactly similar to ROS messages)...

Exercise 2

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

Service Client in Python

../scripts/01_simple_client.py

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

import rospy
from turtlesim.srv import SetPen


if __name__ == '__main__':

    rospy.init_node('I_am_client')

    rospy.wait_for_service('/turtle1/set_pen')

    pen = rospy.ServiceProxy('/turtle1/set_pen', SetPen)

    response = pen(255,0,0,10,0)
```

Service Client in Python

../scripts/02_simple_client.py

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

import rospy
from turtlesim.srv import SetPen

if __name__ == ’__main__’:

    rospy.init_node(’I_am_client’)
    rospy.wait_for_service(’/turtle1/set_pen’)
    pen = rospy.ServiceProxy(’/turtle1/set_pen’, SetPen)

    try:
        pen(255,0,0,10,0)
    except rospy.ServiceException, error:
        print "ops! call has failed with this error: ", error
```

Exercise 3

Service Client in Python

ServiceProxy class

```
rospy.ServiceProxy(
name,
service_class,
persistent=False,
headers=None
)
```

- Let’s check this class definition in rospy! ServiceProxy class
- there is a `wait_for_service` method in the class, let’s try it!

Service Client in Python

../scripts/03_simple_client.py

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

import rospy
from turtlesim.srv import SetPen
from time import sleep

if __name__ == '__main__':

    rospy.init_node('I_am_client')
    pen = rospy.ServiceProxy('/turtle1/set_pen', SetPen)

    pen.wait_for_service()

    try:
        pen(255,0,0,10,0)
    except rospy.ServiceException, error:
        print "ops! call has failed with this error: ", error
```

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

Service Server in Python

- Let's write a service server now!
- The service file must be defined first, we can use:
  - **existing** service files defined in other packages.
  - write our own **custom** service file.
- The process is the same, but writing/using a custom `.srv` file will be covered next section.
  (we already saw how the file looks like, but we need to see how to build the services with Catkin)

Service Server in Python

../scripts/04_simple_server.py

```
#!/usr/bin/env python
import rospy
from std_srvs.srv import SetBool, SetBoolResponse

def destroy(req):
    print "Roger that!.."
    response = SetBoolResponse()
    response.success = True
    if req.data:
        response.message = "You are victorious!"
    else:
        response.message = "enemy spared!"
    return response

if __name__ == '__main__':
    rospy.init_node('red_alert_server')
    print("Ready for melt-down!")
    serv = rospy.Service('destroy_enemy', SetBool, destroy)
    serv.spin()
```

Service Server in Python

Service class

```
rospy.Service(
name,
service_class,
handler,
buff_size=65536,
error_handler=None
)
```

- Let’s check this class definition in rospy! Service class

Exercise 4

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

Using a Custom Service

- We already have seen how to write an `.srv` file.
- Let’s write an `.srv` file for our "useless" node, and tell Catkin to build it.

Using a Custom Service

our custom .srv file

```
string    str
---
string    str
```

- To tell Catkin to build our service, we (again) need to modify `CMakeList.txt` and `package.xml` files of our package.
- It’s similar to what we did for custom messages.

Using a Custom Service

package.xml changes

The changes that we should make to `package.xml`:

1. Add dependencies on message generation:

```
<build_depend> message_generation </build_depend>

<build_depend> message_runtime </build_depend>
```

Using a Custom Service

CMakeList.txt changes

The changes that we should make to `CMakeList.txt`:

1. add `message_generation` to `find_package()` under `COMPONENTS`.
2. add `message_runtime` to `catkin_package()` under `CATKIN_DEPENDS`.
3. add service file(s): (our `.srv` file should be in the `srv` folder in package directory, else it will cause an error):

```
add_service_files(
FILES
ourFile.srv
)
```

Using a Custom Service

CMakeList.txt changes

4. generate messages, and add all dependencies you used in `srv` file:

```
generate_messages(
DEPENDENCIES
std_msgs
)
```

- Note: `add_service_files()` and `generate_messages()` need to be called before `catkin_packge()`, else will error.

Exercise 5

Exercise 6

1. Recap

2. ROS Services

2.1 TurtleSim Services

2.2 Service Description Files

2.3 Service Client in Python

2.4 Service Server in Python

2.5 Using a Custom Service

3. References

References

1. rospy full documentation.
   `http://docs.ros.org/kinetic/api/rospy/html/`
2. ROS Wiki / WritingServiceClient(python).
3. MAS `minimal_ros_packages` GitHub repository. (build instructions, exact copy)
4. Catkin Documentation / building_msgs

Thank you

Any questions?