Software Architectures for Robotics

Lab Session 3

Services, Clients & Parameters

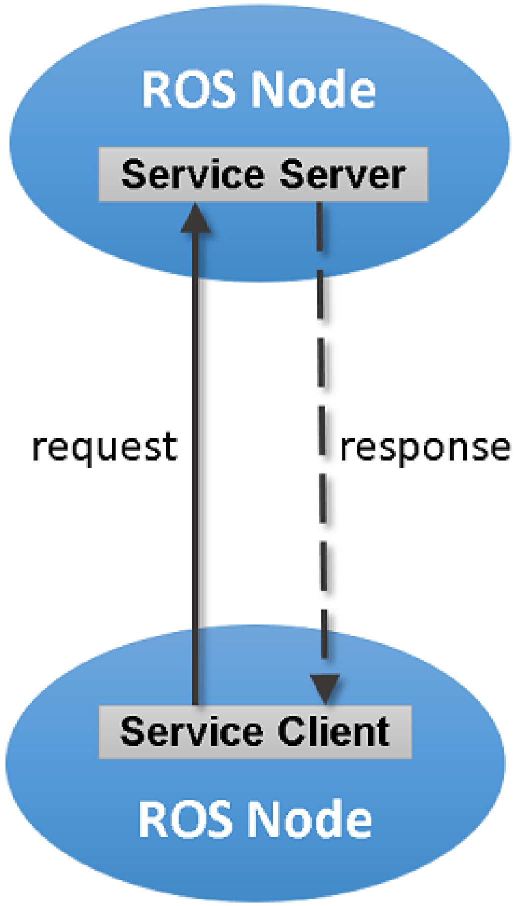
Table of contents

Writing a Service

- When we want some capability to be available to any node in our architecture we can use service nodes.
- A service can process only a request at time.
- A call to a service is always blocking.

Think of it like a sort of C++ function…

Writing a Service



Service Name:	**/example_service**
Service Type:	**roscpp_tutorials/TwoInts**
Request Type:	**roscpp_tutorials/TwoIntsRequest**
Response Type:	**roscpp_tutorials/TwoIntsResponse**

Note: you define a single `.srv` file, but two objects are actually generated!

One for the request and one for the response.

Writing a Service

Let’s start again from the tutorial page!

http://wiki.ros.org/ROS/Tutorials

Let’s look for a tutorial on services and clients…

http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29

Writing a Service

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"


int main(int argc, char **argv)
{
     ros::init(argc, argv, "add_two_ints_server");
     ros::NodeHandle n;

     ros::ServiceServer service = n.advertiseService("add_two_ints", add);
     ROS_INFO("Ready to add two ints.");
     ros::spin();

     return 0;
}
```

Writing a Service

```
bool add(beginner_tutorials::AddTwoInts::Request &req,
beginner_tutorials::AddTwoInts::Response &res)
{
     res.sum = req.a + req.b;
     ROS_INFO("request: x=%ld, y=%ld", (long int)req.a, (long int)req.b);
     ROS_INFO("sending back response: [%ld]", (long int)res.sum);
     return true;
}
```

Writing a Client

```
#include "ros/ros.h"
#include "beginner_tutorials/AddTwoInts.h"
#include <cstdlib>

int main(int argc, char **argv){
     ros::init(argc, argv, "add_two_ints_client");
     ros::NodeHandle n;

     ros::ServiceClient client =
     n.serviceClient<beginner_tutorials::AddTwoInts>("add_two_ints");

     beginner_tutorials::AddTwoInts srv;
     srv.request.a = atoll(argv[1]); srv.request.b = atoll(argv[2]);

     if (client.call(srv)) ROS_INFO("Sum: %ld", (long int)srv.response.sum);
     else ROS_ERROR("Failed to call service add_two_ints");
     return 0;
}
```

Writing a Client

Time to test!

```
$ rosrun beginner_tutorials add_two_ints_server
```

```
$ rosrun beginner_tutorials add_two_ints_client 1 3
```

Output:

```
Requesting 1+3 1 + 3 = 4
```

Topics vs. Services

Topics

- Asynchronous
- Non-Blocking
- No Response

Ideal for data acquisition and signals propagation

Services

- Synchronous
- Blocking
- Gets Response

Ideal for processing and task execution

The Parameters Server

- Use ROS parameters for (shared) configuration values
- Parameters should be rarely modified at runtime
- Recommended way to pass parameters to a node

Avoid using parameters for inter-process communication!

Check `$ rosparam`

Launchfiles

The `roslaunch` package[1] contains tools to read and execute specific `.launch/XML` files[2].

We use such files to:

- Launch a large number of nodes together
- Set necessary parameters
- Include other launchfiles

This is very useful in **large architectures!**

Explore:
http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch

Launchfiles

```
<launch>
    <!-- This is a comment -->
    <!-- How to launch a node-->
    <node name="talker" pkg="beginner_tutorials" type="talker" />
    <!-- output="screen" enables terminal output -->
    <node name="listener" pkg="beginner_tutorials" type="listener" output="screen"/>

    <!-- How to launch a node / alternative -->
    <node pkg="rospy_tutorials" type="talker" name="talker">
        <!-- set a private parameter for the node -->
        <param name="talker_1_param" value="a value" />
        <!-- remap name -->
        <remap from="chatter" to="hello-1"/>
    </node>

    <!-- set a global parameter -->
    <param name="someinteger1" value="1" type="int" />
    <!-- include another launchfile -->
    <include file="$(find some_package)/other.launch" />
    <!-- $(find some_package) evaluates to the package folder -->
</launch>
```

Launchfiles

To launch a launchfile:

```
$ roslaunch [package_name] [file.launch]
```

Or:

```
$ roslaunch [full/path/to/file.launch]
```

`roslaunch` will **automatically** launch a `roscore` if none is running!

Launchfiles

Many packages comes with a launchfile!

Launchfiles are usually located in the `launch` folder.

By default, nodes launched by a launchfile **do not** print anything on the terminal.

Output to terminal must be enabled by `output="screen"` in the `.xml` file.

Rosbag

`rosbag` can be used to record *bags*[3].

Bags are records of messages published on a topic.

Bags can be **recorded** and **republished.**

We can also inspect them, summarize the content or compress them...

Rosbag

To record:

```
$ rosbag record [topic_1] [topic_2] … [topic_n]
$ rosbag record -a
```

To play:

```
$ rosbag play [bagfile.bag]
```

Other commands:

`info, check, fix, compress, filter, ...`

Rosbag

Try it out with `beginner_tutorials` publish and subscribe:

```
$ rosbag record chatter
```

```
$ rosrun beginner_tutorials talker
```

Stop both processes after a while.
A `.bag` file should be in your home now. Now run:

```
$ rosbag play your_file.bag
```

```
$ rosrun beginner_tutorials listener
```

Next Time...

- Debugging and visualization
- ...