академия больших данных

mail.ru group

girafe ai

Introduction to Mobile Robotics course, Seminar 3

Services, Actions, Parameter server and roslaunch

Vladislav Goncharenko, Fall 2021

Materials by Oleg Shipitko

Outline

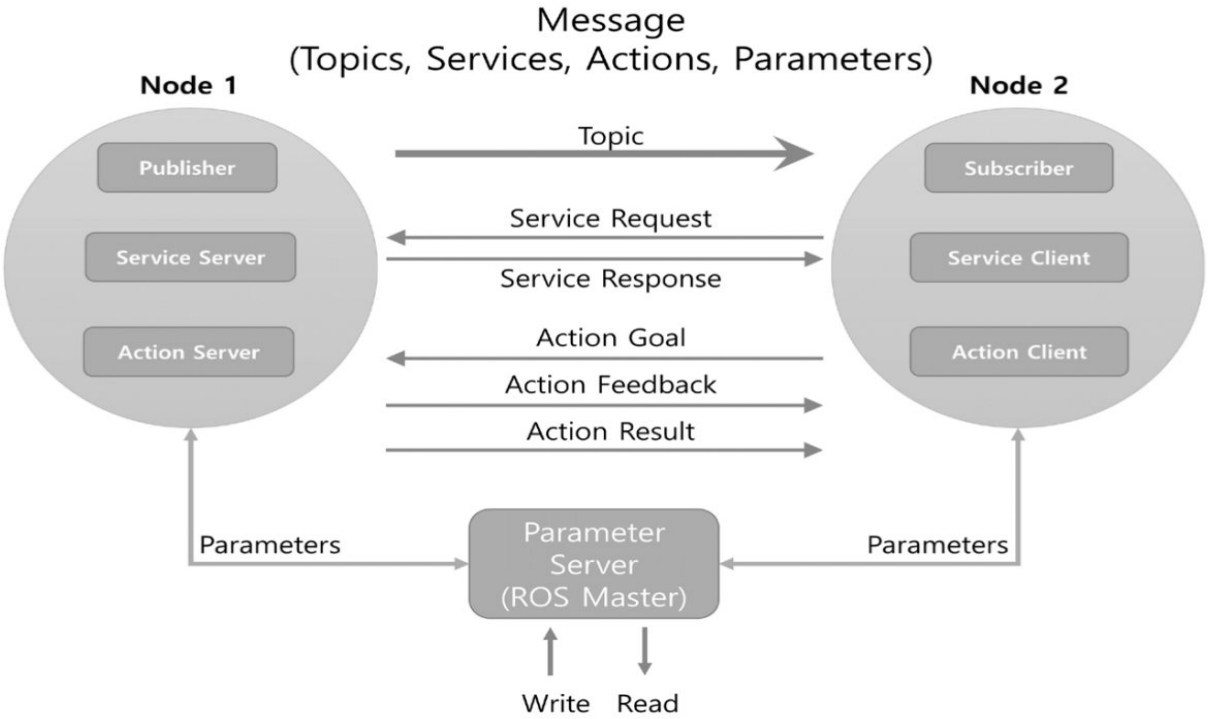
1. Services
2. Actions
3. Parameter server
4. roslaunch

Services

girafe
ai

01

ROS communication types

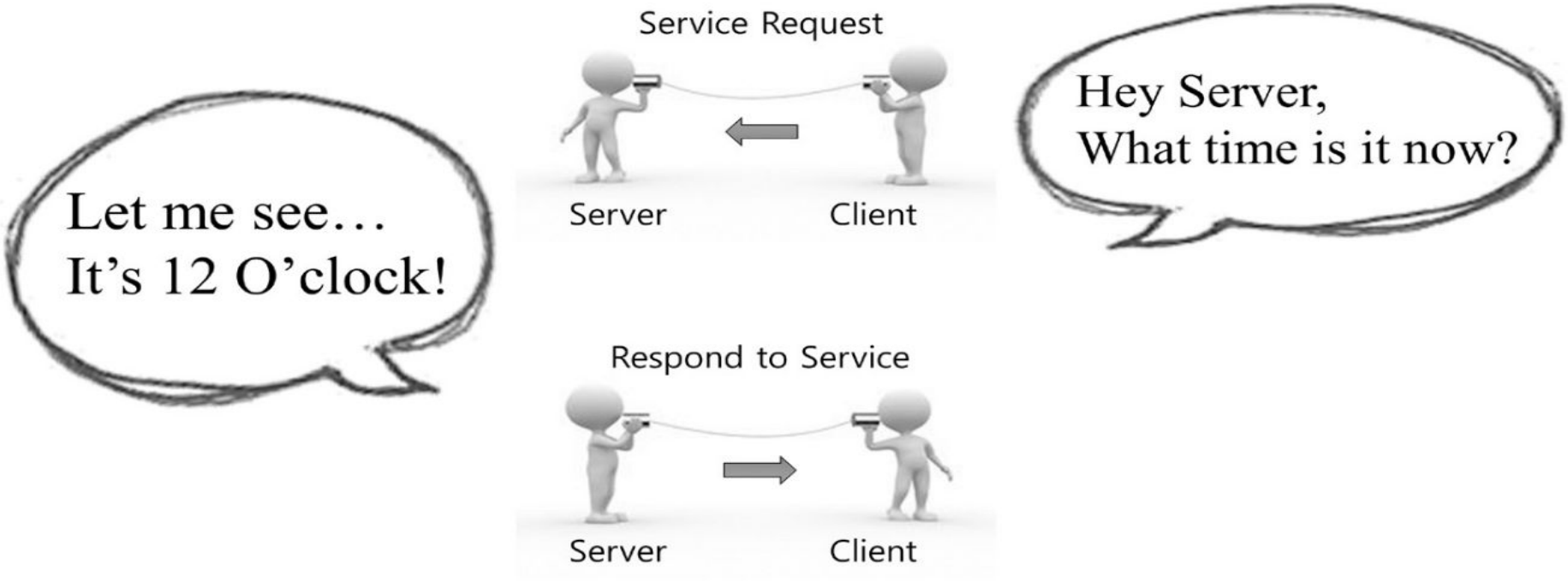


Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

TYPES OF COMMUNICATION

Type	Features	Use cases
Topic	Asynchronous, unidirectional	Continuous data streams
Service	Synchronous, bidirectional	Request-reply with a fast response
Action	Asynchronous, bidirectional	If Service is too long to response, or if you need a feedback in process

Services



Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

ROS INSTALLATION DIRECTORY

http://wiki.ros.org/rospy/Overview/Services

- std_srvs packet contains standard services
- Service definition contains:
  - Request
  - Response
- Empty Request/Response allowed
- Request/Response can be any type:
  - Built-in type (float64)
  - Existing message (`geometry_msgs/Quaternion`)
  - Fixed or dynamic array (`float64[]` or `float64[9]`)

std_srvs/SetBool

```
bool data
# e.g. for hardware enabling / disabling
---
bool success
# indicate successful run of triggered service
string message
# informational, e.g. for error messages
```

std_srvs/Empty

```
---
```

Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

WRITING A SERVICE SERVER

- Import service and it's response from packet:

```
from test_package.srv import GetWindowMedian,GetWindowMedianResponse
from <package>.srv import <Service>,<Service>Response
```

- Create service server:

```
rospy.Service("get_median", GetWindowMedian, handle_get_median)
rospy.Service(name, service_class, handler, buff_size=65536, error_handler=None)
```

- Define callback function:

```
def handle_get_median(req):
    # some service-handling code
    return GetWindowMedianResponse(<response_data>)
```

WRITING A SERVICE CLIENT

- Block program until the is no connection to service:

```
rospy.wait_for_service("get_median")
rospy.wait_for_service(service, timeout=None)
```

- Create service client:

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

- Send request to service:

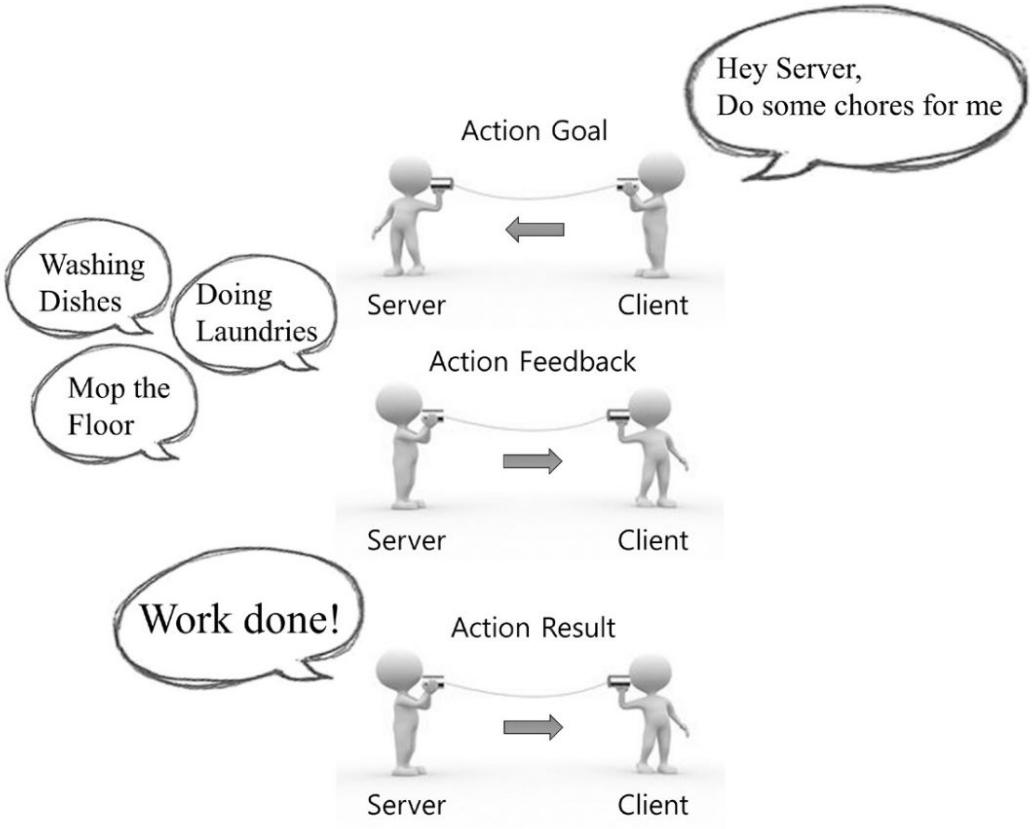
```
response = get_median(<request_data>)
```

Actions

girafe
ai

02

ACTIONS

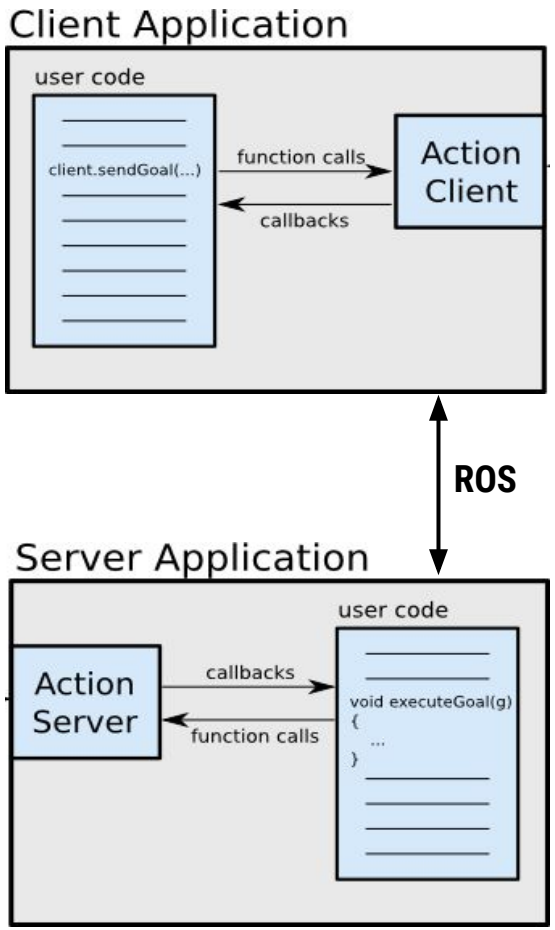


Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

ACTIONS

http://wiki.ros.org/actionlib

- actionlib packet provides an API for client-server calls for actions
- Actions made of three parts:
  - Goal
  - Feedback
  - Result
- Every part can contain any amount of fields of any type:
  - Built-in type (float64)
  - Existing message (`geometry_msgs/Quaternion`)
  - Fixed or dynamic array (`float64[]` or `float64[9]`)



ACTION FILE

http://wiki.ros.org/actionlib

- Saved in the `/action` directory of the packet
- Requires `actionlib_msgs` dependency in `CmakeLists.txt` и `package.xml` (as `message_generation` dependency for custom messages)
- It used to generate messages that actions use internally for communication between server and client:
  - DoDishesAction.msg
  - DoDishesActionGoal.msg
  - DoDishesActionResult.msg
  - DoDishesActionFeedback.msg
  - DoDishesGoal.msg
  - DoDishesResult.msg
  - DoDishesFeedback.msg

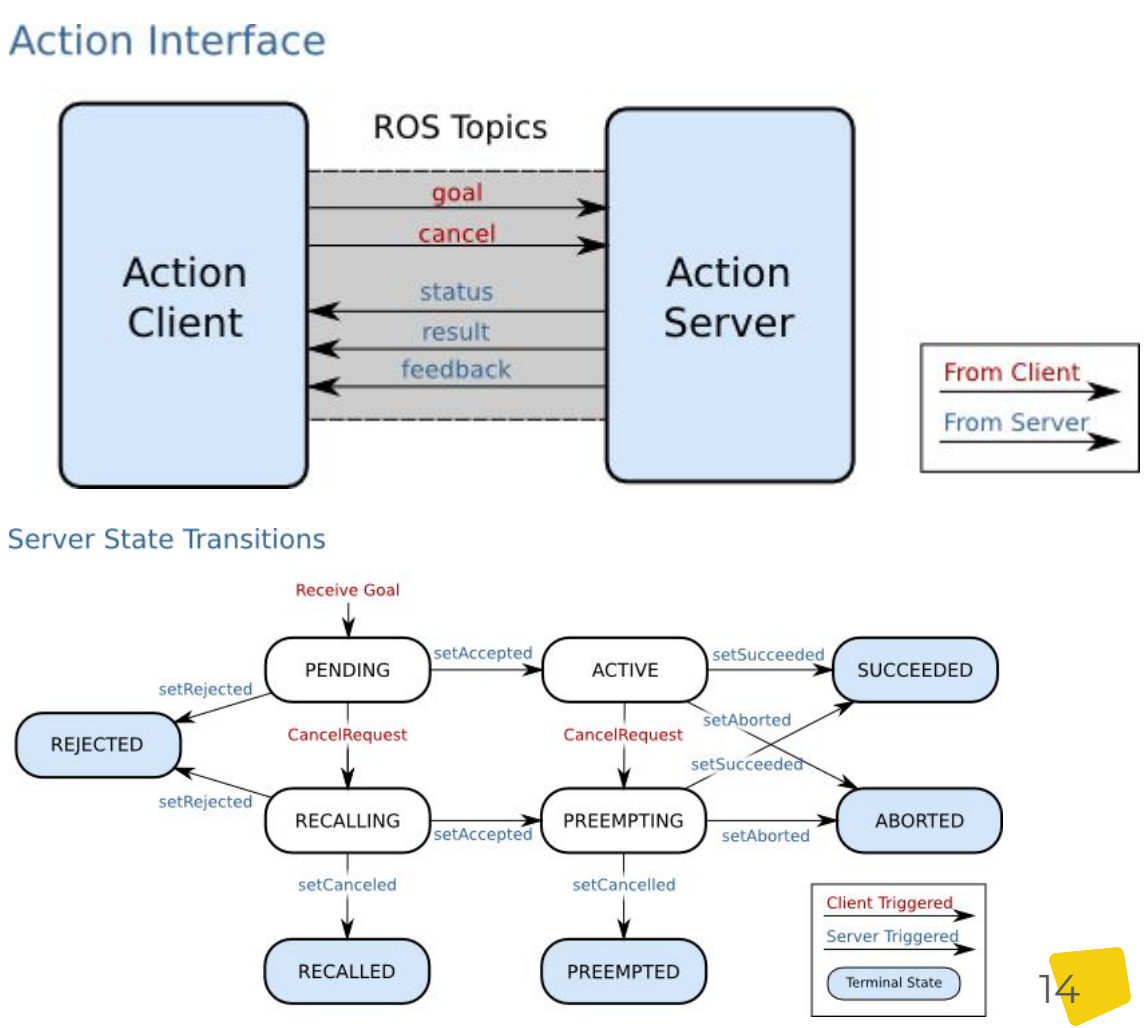
./action/DoDishes.action

```
# Define the goal
uint32 dishwasher_id
# specify what dishwasher we want
to use
---
# Define the result
uint32 total_dishes_cleaned
---
# Define the feedback
float32 percent_complete
```

ACTIONS

http://wiki.ros.org/actionlib/DetailedDescription

- **goal** - used to send data to action
- **cancel** - for cancelling action
- **status** - for getting current status of action (possible states)
- **feedback** - sends information from server to client during action
- **result** - sends result of the action only once



Parameter server

girafe ai

03

PARAMETER SERVER

http://wiki.ros.org/Parameter%20Server

Parameter Server – set of parameters, which is accessible by any node in the system. Used to store different parameters and provides access to them in a realtime. Runs as part of ***rosmaster***.

Data types allowed by ***Parameter Server*** :

- ❏ 32-bit integers
- ❏ booleans
- ❏ strings
- ❏ doubles
- ❏ iso8601 dates
- ❏ lists
- ❏ base64-encoded binary data

Parameters can be accessed with a client libraries (***roscpp***, ***rospy***, ...) and using CLI tool ***rosparam***.

Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

PARAMETER SERVER

http://wiki.ros.org/Parameter%20Server

- Getting parameters from Python code

```
global_name = rospy.get_param("/global_name")
relative_name = rospy.get_param("relative_name")
private_param = rospy.get_param('~private_name')
default_param = rospy.get_param('default_param',
                                'default_value')
# fetch a group (dictionary) of parameters
gains = rospy.get_param('gains')
p, i, d = gains['p'], gains['i'], gains['d']
```

- Setting parameters from Python code

```
# Using rospy and raw python objects
rospy.set_param('a_string', 'baz')
rospy.set_param('~private_int', 2)
rospy.set_param('list_of_floats', [1., 2., 3., 4.])
rospy.set_param('bool_True', True)
rospy.set_param('gains', {'p': 1, 'i': 2, 'd': 3})
```

- Looking for a parameter and deleting it

```
if rospy.has_param('to_delete'):
   rospy.delete_param('to_delete')
```


Source: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim

roslaunch

girafe
ai

04

ROSLAUNCH

http://wiki.ros.org/roslaunch
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

- **Problem**: in a process of development complex systems often emerges a need to run many nodes, set their parameters and even select special host to run node(in case of distributed system).

ROSLAUNCH

http://wiki.ros.org/roslaunch
http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

- **Problem**: in a process of development complex systems often emerges a need to run many nodes, set their parameters and even select special host to run node(in case of distributed system).
- **Solution**: roslaunch - CLI tool which lets to define startup process of the system with `xml` files (with `.launch` extension) and run whole system with one command.
  - Automatically runs `roscore`
  - Command `roslaunch_add_file_check(launch)` in `CMakeLists.txt` allows to test launch file for typical syntax errors

ROSLAUNCH

```
<launch>
 <!-- local machine already has a definition by default.This tag overrides the default definition
with specific ROS_ROOT and ROS_PACKAGE_PATH values -->
 <machine name="local_alt" address="localhost" default="true" ros-root="/u/user/ros/ros/"
ros-package-path="/u/user/ros/ros-pkg" />
 <!-- a basic listener node -->
 <node name="listener-1" pkg="rospy_tutorials" type="listener" />
 <!-- pass args to the listener node -->
 <node name="listener-2" pkg="rospy_tutorials" type="listener" args="-foo arg2" />
 <!-- a respawn-able listener node -->
 <node name="listener-3" pkg="rospy_tutorials" type="listener" respawn="true" />
 <!-- start listener node in the 'wg1' namespace -->
 <node ns="wg1" name="listener-wg1" pkg="rospy_tutorials" type="listener" respawn="true" />
 <!-- start a group of nodes in the 'wg2' namespace -->
 <group ns="wg2">
   <!-- remap applies to all future statements in this scope. -->
   <remap from="chatter" to="hello"/>
   <node pkg="rospy_tutorials" type="listener" name="listener" args="--test" respawn="true" />
   <node pkg="rospy_tutorials" type="talker" name="talker">
     <!-- set a private parameter for the node -->
     <param name="talker_1_param" value="a value" />
     <!-- nodes can have their own remap args -->
     <remap from="chatter" to="hello-1"/>
     <!-- you can set environment variables for a node -->
     <env name="ENV_EXAMPLE" value="some value" />
   </node>
 </group>
</launch>
```

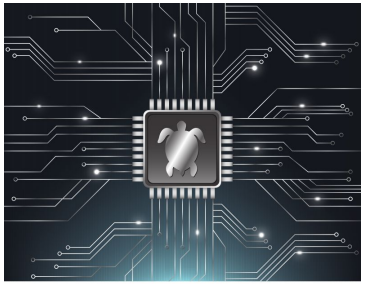
ROSLAUNCH TAGS

http://wiki.ros.org/roslaunch

- `<launch>` – root tag. Obligatory for any .launch file
- `<node>` – tag for starting a node.
- `<machine>` – defines a host which will run a node. Not used in case of local run
- `<include>` – allows to include external .xml file to the current one
- `<remap>` – remaps arguments
- `<env>` – sets environment variables
- `<param>` – sets parameter value into `parameter server`
- `<rosparam>` – sets parameter value into `parameter server` from the .yaml file
- `<group>` – applies same configuration to several nodes(ex., set of namespace).
- `<test>` – same as <node>, but implies running a node to test other nodes
- `<arg>` – set running arguments

ADDITIONAL RESOURCES

1. Book: ROS Robot Programming. YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim
2. ROS Officiel Tutorials
3. Clearpath Robotics ROS Tutorial
4. The history of ROS creation



Thanks for attention!

Questions? Additions? Welcome!

---

girafe
ai