Hochschule
Bonn-Rhein-Siegg
University of Applied Sciences

b-it Bonn-Aachen
International Center for
Information Technology

ROS Actions

Foundation Course

March 19, 2020

Hassan Umari

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

Recap

Summary of yesterday's session

We saw that ROS services allow for **two-way communication** between nodes. However:

1. Client **must wait** for the response: a call to a service blocks the code until the execution of the service is finished.
2. During execution of the service, the client cannot ask the server to cancel the request; **services are not preemptable**.
3. During execution of the service, there is **no** way for the server to send **feedback** to the client.

Recap

Summary of yesterday's session

1. **ROS actions** are there to solve these limitations.
2. They are suitable for long-running tasks, which can be interrupted (canceled).
3. Popular case example: sending a pose to the robot, where the action server is responsible for controlling the robot to reach the goal pose.

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 Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

ROS Actions

Actions:

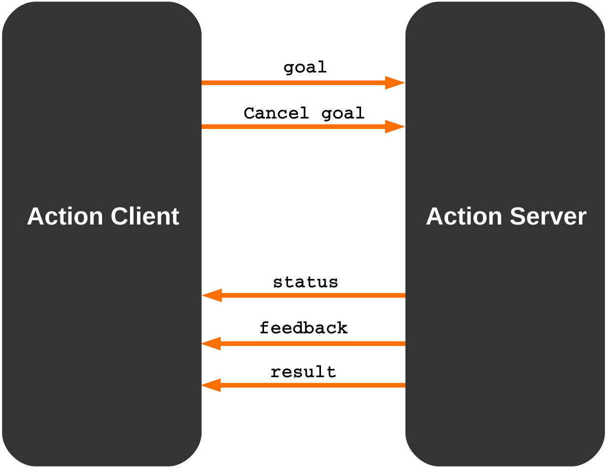
- Communication happens between two nodes, the action **server** node, and the action **client** node.
- A client node sends a **goal** to the action server. The client does NOT have to wait for the response.
- The server node executes the action, during execution it can optionally send feedback messages to the client.

ROS Actions

Actions:

- Meanwhile, the client can receive **feedback** messages from the server, it can also check on the **status** of the current goal, or can even ask the server to **cancel** the action.
- Once the server has finished executing the action, it sends a **result** message back to the client.

ROS Actions



extracted and edited from: http://wiki.ros.org/actionlib/DetailedDescription

ROS Actions

Actions:

- ROS actions are built on top of ROS messages, they are not part of `rospy`.
- They are provided in the `actionlib` ROS stack which is installed by default when you install ROS.

ROS Actions

Actions:

- There are no command-line tools to introspect ROS actions.
  - Example: there is no `rosaction list` command!
  - Since `actionlib` is built on top of ROS messages, an action server exposes the following topics which can be used for introspection:
    - `/<action name>/cancel`
    - `/<action name>/feedback`
    - `/<action name>/goal`
    - `/<action name>/result`
    - `/<action name>/status`

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

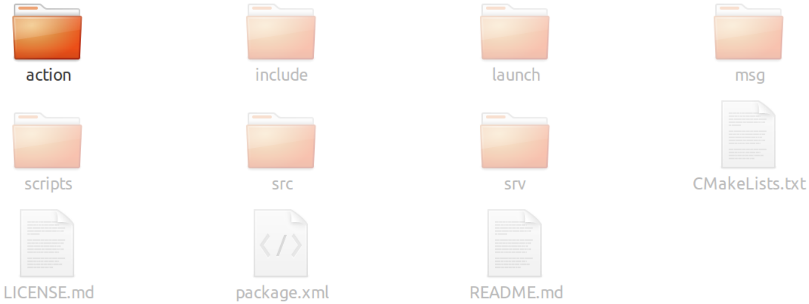
2.4 SimpleActionClient in Python

3. References

Action Description Files

.action file format

- A ROS action is defined in a text file with `.action` extension.
- `.action` files are normally placed (not a must) in `action` folder inside the package.



Action Description files

.action file format

- The file consists of three sections: **goal** message, **feedback** message, and **result** message, each separated with "`---`":

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

Action Description files

.action file format

- Example `.action` file:

```
geometry_msgs/PoseStamped    target_pose
---
---
geometry_msgs/PoseStamped    base_position
```

Action Description files

.action file format

- When you build your package, Catkin reads `.action` file and generates the following ROS messages (below files are generated from `DoDishes.action` file):
  - `DoDishesAction.msg`.
  - `DoDishesActionGoal.msg`.
  - `DoDishesActionResult.msg`.
  - `DoDishesActionFeedback.msg`.
  - `DoDishesGoal.msg`.
  - `DoDishesResult.msg`.
  - `DoDishesFeedback.msg`.
- Let's see how to build a package with action files in the next exercise.

Exercise 1

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

actionlib **overview**

The actionlib stack provides two Python classes (also in C++) which you can use to implement an action server:

- ActionServer class:
  this allows you to define action servers that can accept multiple goals concurrently. (spanning multiple threads)

`actionlib` overview

and..

- `SimpleActionServer` class:
  this class allows you to define action servers that can only handle one goal at a time.

  Quote from `actionlib` documentation:
  "*The SimpleActionServer implements a singe goal policy on top of the ActionServer class.*"

actionlib **overview**

- Throughout this session, we will only use the SimpleActionServer class to implement action servers.
- For action clients, we will use the SimpleActionClient class.

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

SimpleActionServer in Python

../scripts/00_simple_server.py

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

import rospy
from actionlib import SimpleActionServer
from my_third_package.msg import MoveTurtleAction

def action_cb(goal):
    print("moving turtle to pose: ", goal)
    # Write here the logic to control turtle
    print("done")
    server.set_succeeded()

if __name__ == "__main__":
    rospy.init_node("move_turtle")
    server = SimpleActionServer("move_turtle_action",
                                MoveTurtleAction,
                                execute_cb=action_cb)
```

Exercise 2

SimpleActionServer in Python

- In the previous script, the action server does not allow goal cancellation.

- It does not send any feedback as well.

- So it is up to you, the implementer, to make the action server send feedback, make it preemtable, etc..

SimpleActionServer in Python

- Let's now see what might be a better implementation for an action server..
- Check script `01_simple_server.py`

Exercise 3

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

SimpleActionClient in Python

- Simple **SimpleActionClient**:

  Let's check script `02_simple_client.py`

Exercise 4

1. Recap

2. ROS Actions

2.1 Action Description Files

2.2 `actionlib` overview

2.3 SimpleActionServer in Python

2.4 SimpleActionClient in Python

3. References

References

1. actionlib full documentation.
   `https://docs.ros.org/api/actionlib/html/`
2. ROS Wiki / actionlib.
3. MAS `minimal_ros_packages` GitHub repository. (build instructions, exact copy)

Thank you

Any questions?