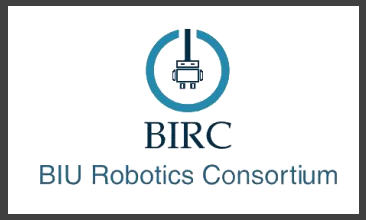
October 2016



ROS – Lecture 8

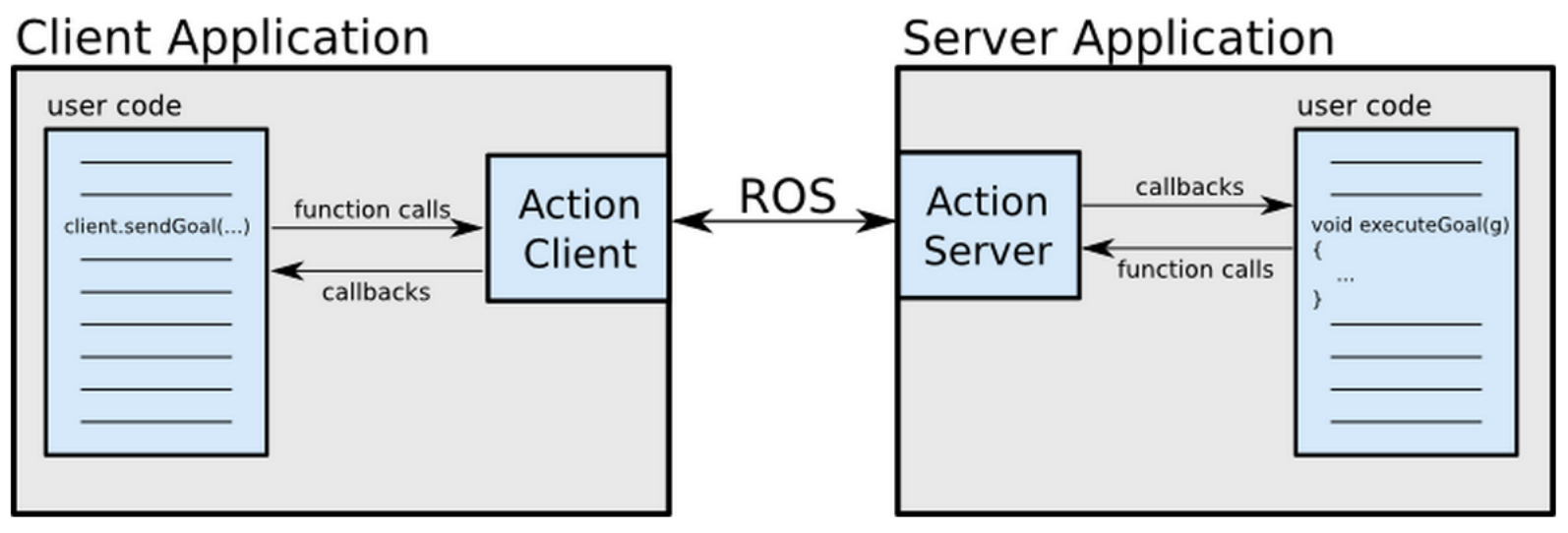
ROS actions
Sending goal commands from code
Making navigation plans

Lecturer: Roi Yehoshua
roiyeho@gmail.com

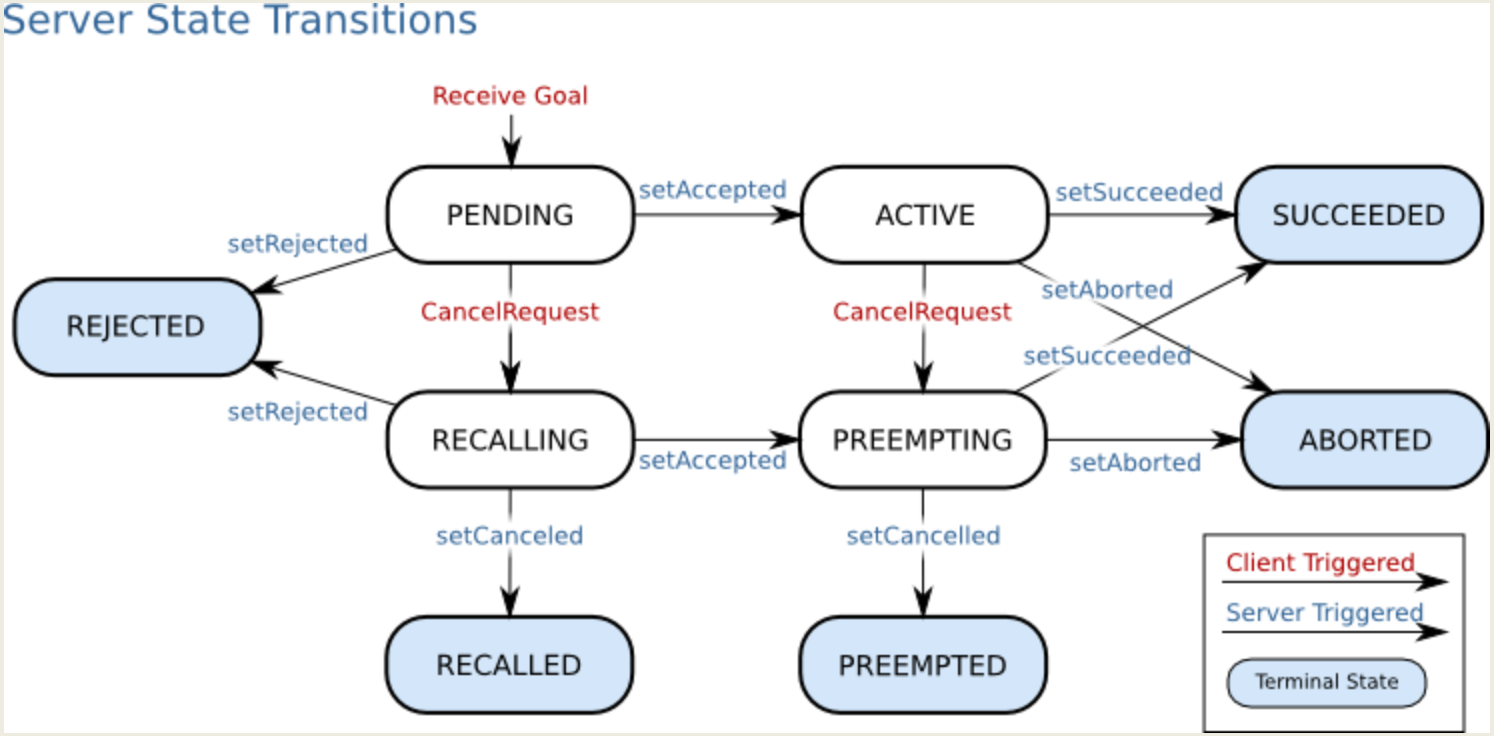
Actions

- http://wiki.ros.org/actionlib
- While services are handy for simple get/set interactions like querying status and managing configuration, they don't work well with long-running tasks
  - such as sending a robot to some distant location
- ROS **actions** are the best way to implement interfaces to time-extended, goal-oriented behaviors
- Similar to the request and response of a service, an action uses a *goal* to initiate a behavior and sends a *result* when the behavior is complete
- But the action further uses feedback to provide updates on the behavior's progress toward the goal and also allows for goals to be canceled
- Actions are asynchronous (in contrast to services)

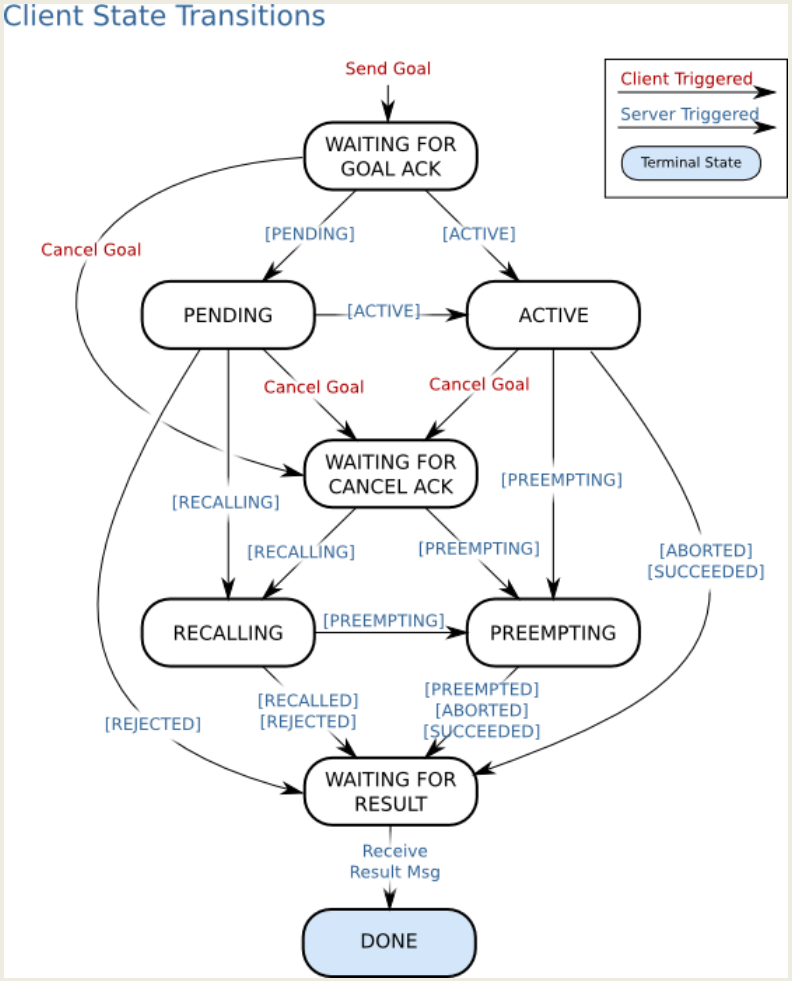
Action Client-Server Interaction



Action Server State Transitions



Action Client State Transitions



.action File

- The action specification is defined in an .action file
  - These files are placed in the package's **./action** directory
- Example for an action file:

```
# Define the goal
uint32 dishwasher_id  # Specify which dishwasher we want to use
---
# Define the result
uint32 total_dishes_cleaned
---
# Define a feedback message
float32 percent_complete
```

.action File

- From the action file the following message types are generated:
  - DoDishesAction.msg
  - DoDishesActionGoal.msg
  - DoDishesActionResult.msg
  - DoDishesActionFeedback.msg
  - DoDishesGoal.msg
  - DoDishesResult.msg
  - DoDishesFeedback.msg
- These messages are then used internally by actionlib to communicate between the ActionClient and ActionServer

SimpleActionClient

- A simple client implementation which supports only one goal at a time
- The action client is templated on the action definition, specifying what message types to communicate to the action server with
- The action client c'tor also takes two arguments:
  - The server name to connect to
  - A boolean option to automatically spin a thread
    - If you prefer not to use threads (and you want actionlib to do the 'thread magic' behind the scenes), this is a good option for you.

SimpleActionClient

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

import actionlib
from basics.msg import TimerAction, TimerGoal, TimerResult

rospy.init_node('timer_action_client')
client = actionlib.SimpleActionClient('timer', TimerAction)
client.wait_for_server()
goal = TimerGoal()
goal.time_to_wait = rospy.Duration.from_sec(5.0)
client.send_goal(goal)
client.wait_for_result()
print('Time elapsed: %f'%(client.get_result().time_elapsed.to_sec()))
```

SendGoals

- The next code is a simple example to send a goal to move the robot from the Python code
- In this case the goal would be a PoseStamped message that contains information about where the robot should move to in the world
- http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals

SendGoals

- First create a new package called **send_goals**
- This package depends on the following packages:
  - actionlib
  - geometry_msgs
  - move_base_msgs

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg send_goals std_msgs rospy roscpp actionlib tf
geometry_msgs move_base_msgs
```