ROS ActionLib

SAPIENZA
UNIVERSITÀ DI ROMA

Roberto Capobianco

Thanks to: Luca Iocchi

Dipartimento di Ingegneria Informatica, Automatica e Gestionale

What is ActionLib

- Node A sends a request to node B to perform some task
- Services are suitable if task is "instantaneous"
- Actions are more adequate when task takes time and we want to monitor, have continuous feedback and possibly cancel the request during execution

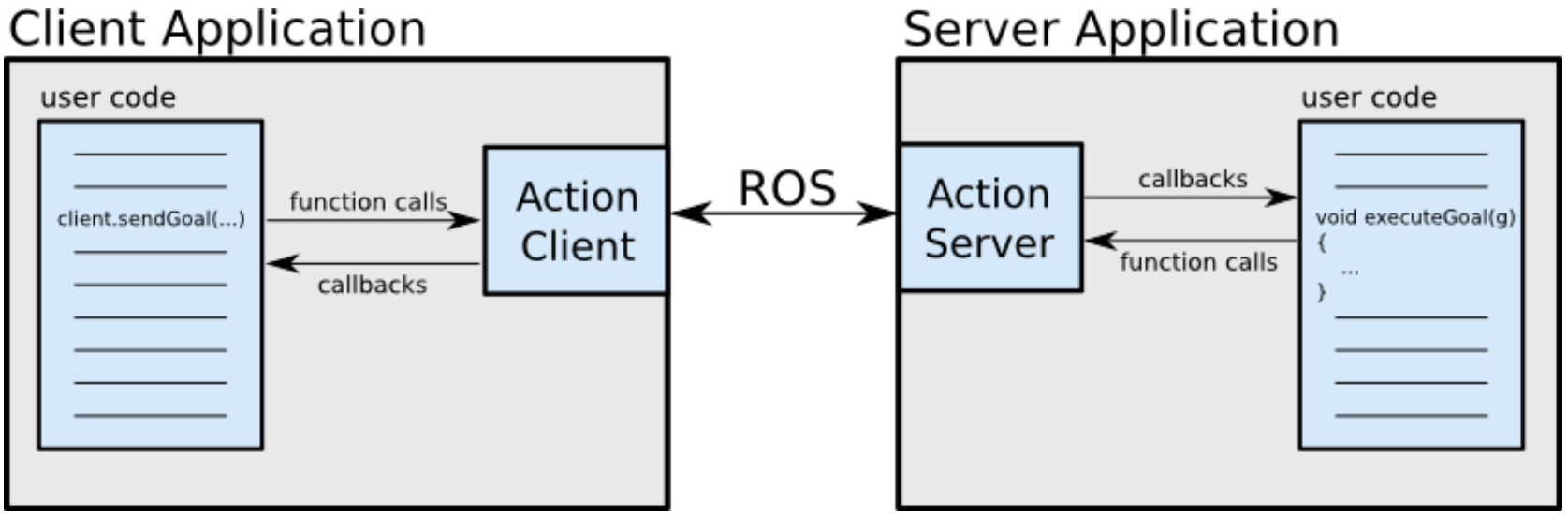
ActionLib

- actionlib is a package that provides tools to
  - creates servers that execute long-running tasks (that can be preempted).
  - creates clients that interact with servers

References

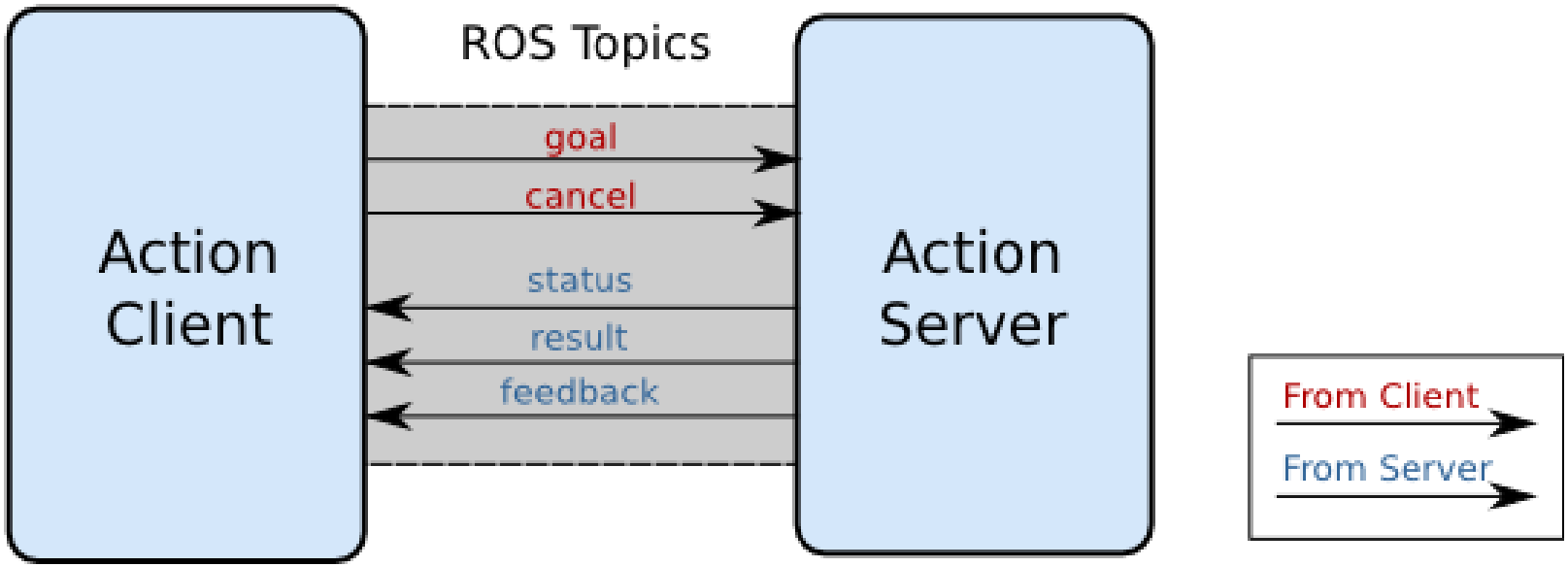
- http://wiki.ros.org/actionlib
- http://wiki.ros.org/actionlib/DetailedDescription
- http://wiki.ros.org/actionlib/Tutorials

ActionLib Schema

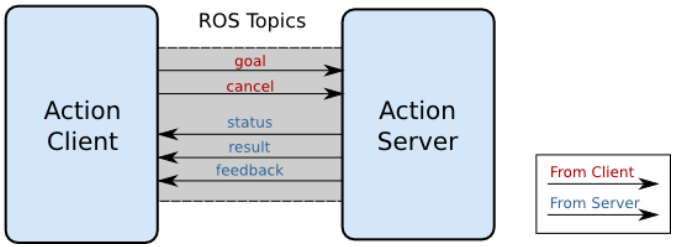


Client-server interaction using "*ROS Action Protocol*"

Client-Server Interaction



Client-Server Interaction

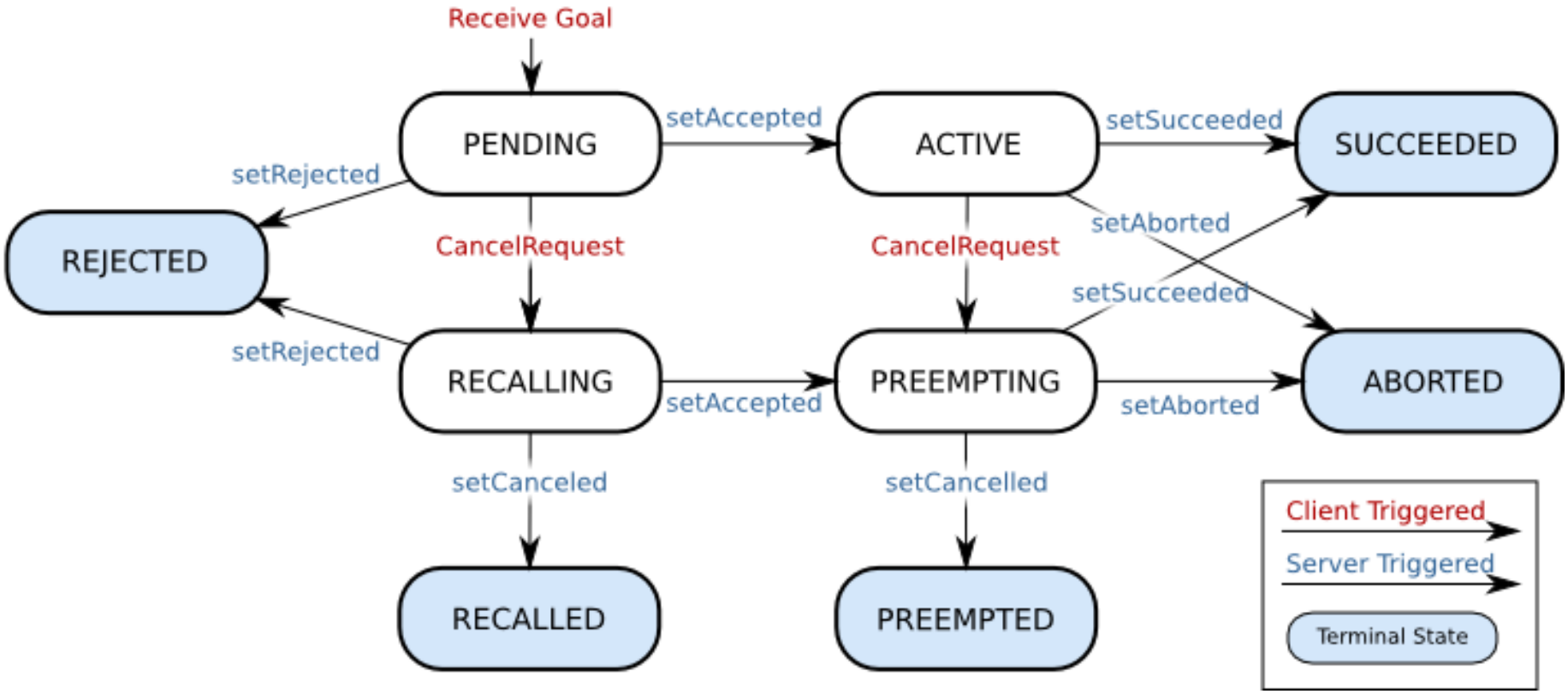


- **goal** - Used to send new goals to server
- **cancel** - Used to send cancel requests to server
- **status** - Used to notify clients on the current state of every goal in the system.
- **feedback** - Used to send clients periodic auxiliary information for a goal
- **result** - Used to send clients one-time auxiliary information upon completion of a goal

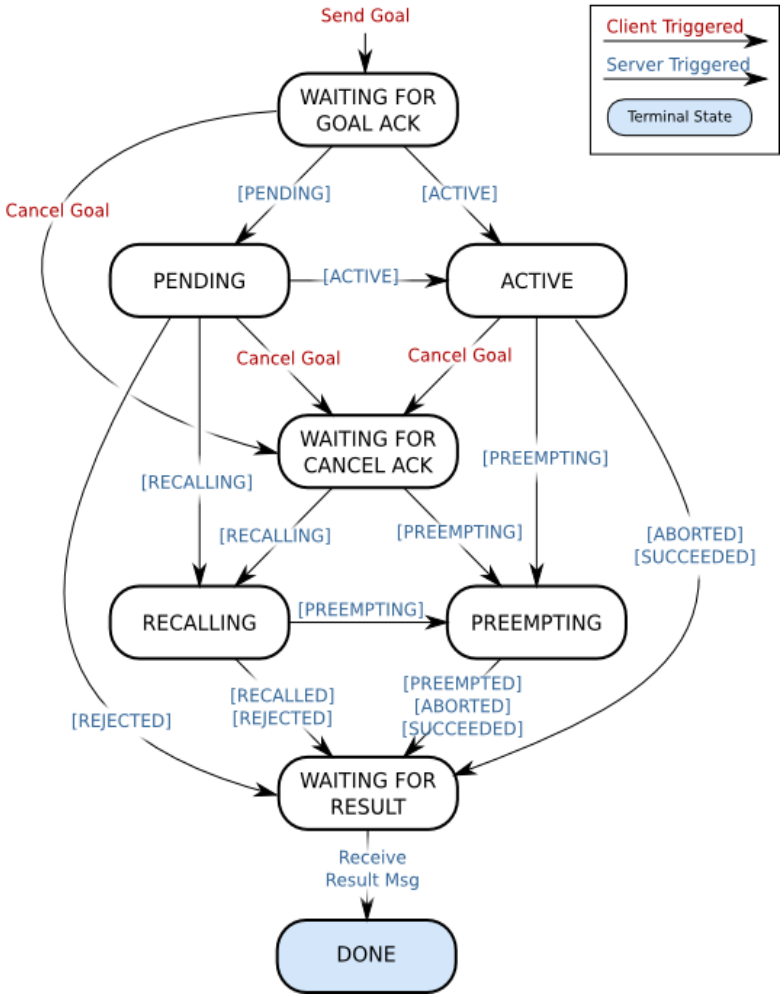
Actions and Goal ID

- Action templates are defined by a name and some additional properties through an .action structure defined in ROS
- Each instance of an action has a unique Goal ID
- Goal ID provides the action server and the action client with a robust way to monitor the execution of a particular instance of an action.

Server State Machine

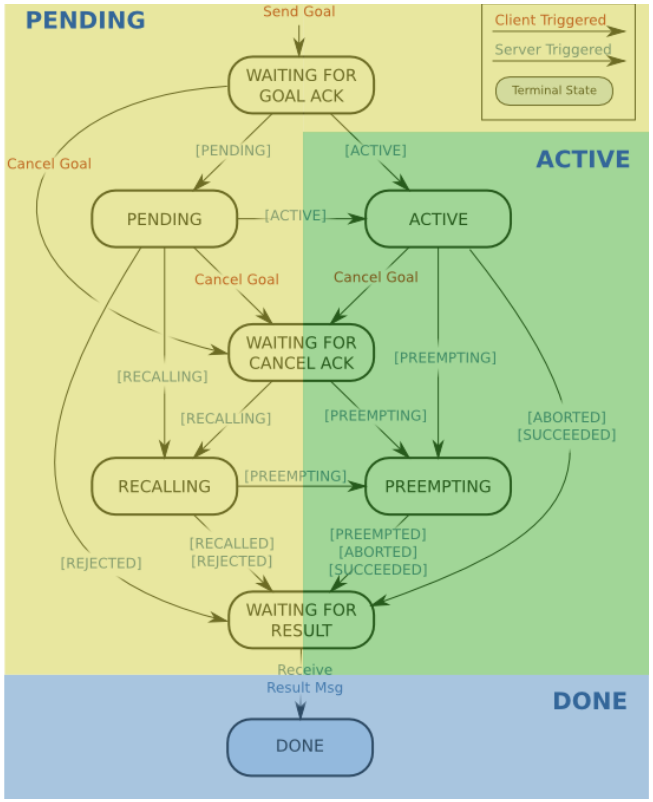


Client State Machine



SimpleActionServer/Client

- SimpleActionServer: implements a single goal policy.
- Only one goal can have an active status at a time.
- New goals preempt previous goals based on the stamp in their GoalID field.
- SimpleActionClient: implements a simplified ActionClient



Example: move_base action server

- **Action Subscribed Topics**
  - move_base/goal (move_base_msgs/MoveBaseActionGoal): A goal for move_base to pursue in the world.
  - move_base/cancel (actionlib_msgs/GoalID): A request to cancel a specific goal.
- **Action Published Topics**
  - move_base/feedback (move_base_msgs/ MoveBaseActionFeedback): Feedback contains the current position of the base in the world.
  - move_base/status (actionlib_msgs/GoalStatusArray): Provides status information on the goals that are sent to the move_base action.
  - move_base/result (move_base_msgs/MoveBaseActionResult): Result is empty for the move_base action.

Sending a goal with move_base

```
typedef
actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
MoveBaseClient;
//tell the action client that we want to spin a thread by default
MoveBaseClient ac("move_base", true);
//wait for the action server to come up
while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Waiting for the move_base action server to come up");
}
 // setting the goal
move_base_msgs::MoveBaseGoal goal;
goal.target_pose.header.frame_id = "base_link";
goal.target_pose.header.stamp = ros::Time::now();
goal.target_pose.pose.position.x = 1.0;
goal.target_pose.pose.orientation.w = 1.0;
```

Sending a goal with move_base

```
// sending the goal
ac.sendGoal(goal);

// wait until finish
while (!ac.waitForResult(ros::Duration(1.0)))
    ROS_INFO("Running...");

// print result
if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    ROS_INFO("Hooray, the base moved 1 meter forward");
else
    ROS_INFO("The base failed to move forward 1 meter for some
reason");
```

Cancelling a goal with move_base

```
typedef
actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction>
MoveBaseClient;

MoveBaseClient ac("move_base", true);
...

// Cancel all active goals
ac.cancelAllGoals();
```

Defining actions

Define an action file (e.g., Turn.action in rp_action/action folder)

#Goal

- specification of the goal

#Result

- specification of the result

#Feedback

- specification of the feedback

```
# Goal
# target_angle [DEG]
float32 target_angle
# flag ABS/REL
string absolute_relative_flag
# max angular velocity [DEG/s]
float32 max_ang_vel
---
# Result
string result
---
# Feedback
string feedback
```

Building actions

Add the following to your CMakeLists.txt file before catkin_package().

```
find_package(catkin REQUIRED genmsg actionlib_msgs actionlib)
add_action_files(DIRECTORY action FILES DoDishes.action)
generate_messages(DEPENDENCIES actionlib_msgs)
```

Additionally, the package's package.xml must include the following dependencies:

```
<build_depend>actionlib</build_depend>
<build_depend>actionlib_msgs</build_depend>
<run_depend>actionlib</run_depend>
<run_depend>actionlib_msgs</run_depend>
```

Writing an action server

```
class TurnActionServer {
protected:
   ros::NodeHandle nh;
   std::string action_name;
   actionlib::SimpleActionServer<rp_actions::TurnAction> turn_server;
public:
   TurnActionServer(std::string name) :        action_name(name),
      turn_server(nh, action_name,
                                    boost::bind(&TurnActionServer::executeCB, this, _1), false)
   {  turn_server.start();  }

   void executeCB(const rp_actions::TurnGoalConstPtr& goal)  {
      ...
      }
}
```

Writing an action client

```
std::string action_name = "turn";
// Define the action client (true: we want to spin a thread)
actionlib::SimpleActionClient<rp_actions::TurnAction> ac(action_name , true);

// Wait for the action server to come up
while(!ac.waitForServer(ros::Duration(5.0))) {
    ROS_INFO("Waiting for turn action server to come up");
}

// Set the goal
rp_actions::TurnGoal goal;
goal.target_angle = 90;  // target deg
goal.absolute_relative_flag = "REL"; // relative
goal.max_ang_vel = 45.0;  // deg/s

// Send the goal
ac.sendGoal(goal);
```

ActionServer/Client

- **ActionServer** and **ActionClient** use the complete set of states and transitions.
- More difficult to program.
- Needed when we want to execute multiple instances of an action at the same time (parallel actions).
- Implemented in PNPros module.

Conclusions

- **ActionLib** powerful library to write and control duration processes/actions
- SimpleActionServer/Client easy to use, standard ActionServer/Client more difficult, but not typically needed
- ActionLib is integrated with other libraries for action combination:
- - SMACH: hierarchical state machines
- http://wiki.ros.org/smach
- - **PNP: Petri Net Plans**
- http://pnp.dis.uniroma1.it

Homework

Write an action for time countdown.
Write a SimpleActionServer that counts down for *n* seconds, displaying on the screen the count down at each second.
Write a SimpleActionClient that activates a count down specifying the amount of seconds
Write a SimpleActionClient that stops the count down

Note: with SimpleActionServer/Client it is not possible to run two counters at the same time