ACTIONLIB

ROBOTICS

POLITECNICO
MILANO 1863

WHAT IS ACTIONLIB

Node A sends a request to node B to perform some task

Service	**Action**
Small execution time	Long execution time
Requesting node can wait	Requesting node cannot wait
No status	Status monitoring
No cancellation	Cancellation

WHAT IS ACTIONLIB

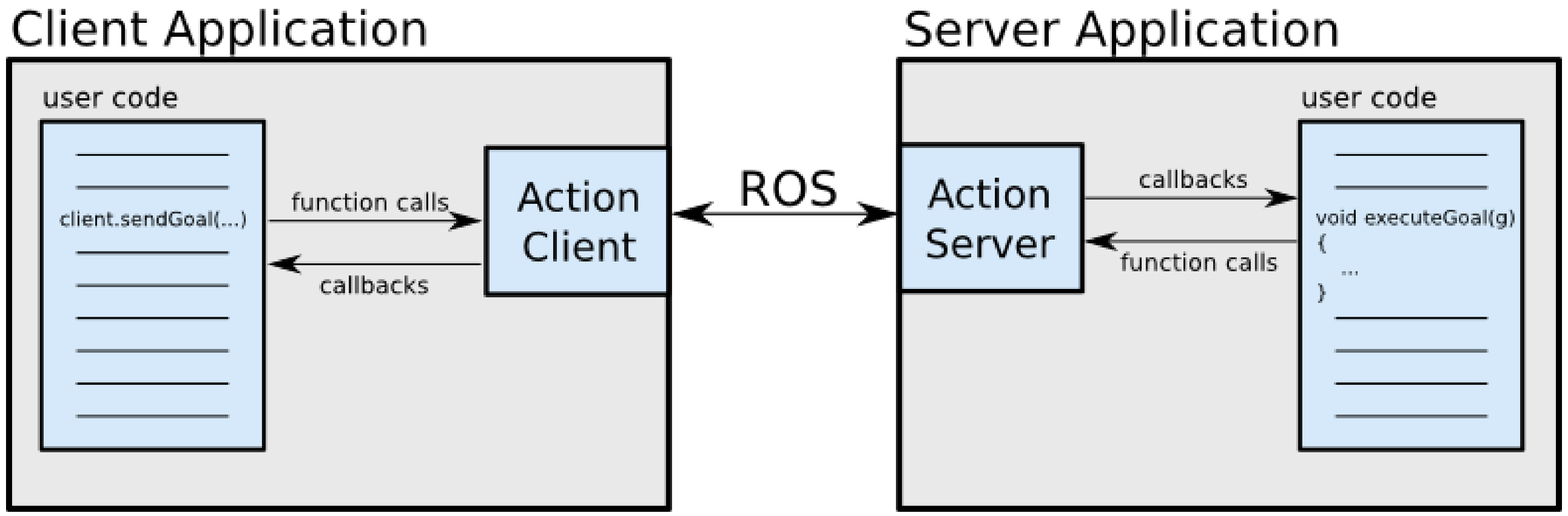
actionlib package is:

- sort of ROS implementation of threads
- based on a client/server paradigm

And provides tools to:

- create servers that execute long-running tasks (that can be preempted).
- create clients that interact with servers

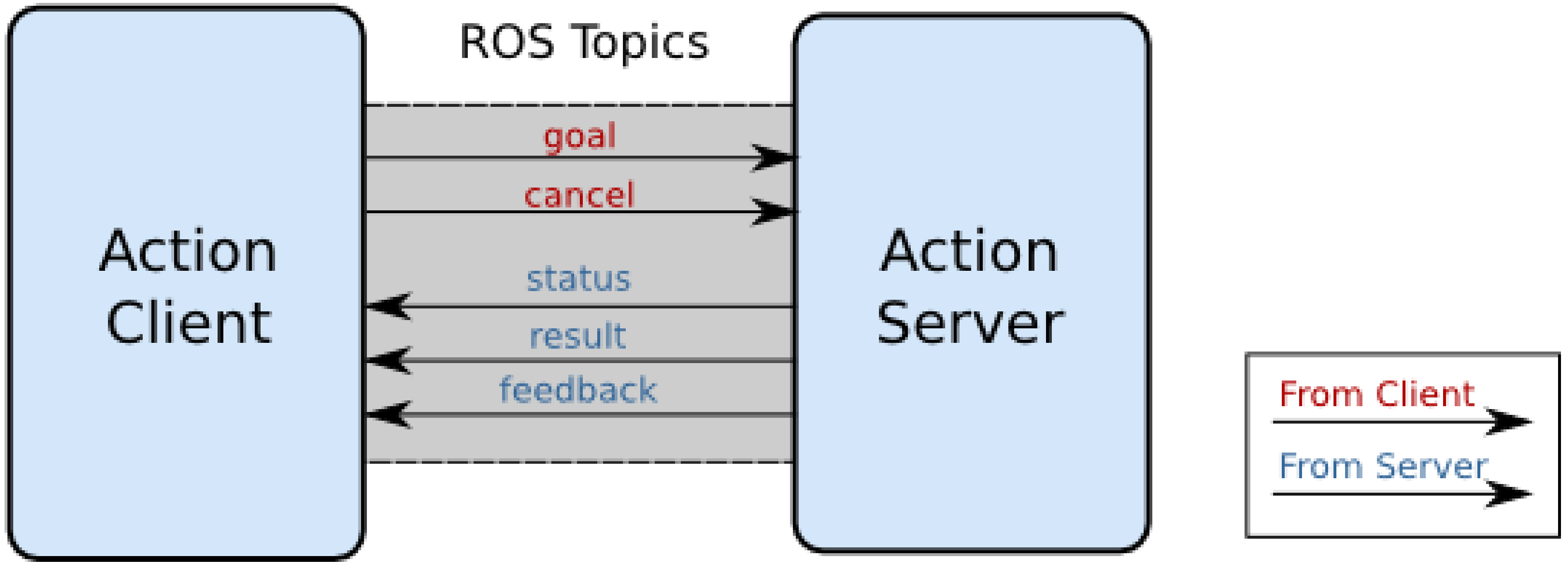
WHAT IS ACTIONLIB



The ActionClient and ActionServer communicate via a "ROS Action Protocol", which is built on top of ROS messages

CLIENT-SERVER INTERACTION

Action Interface



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

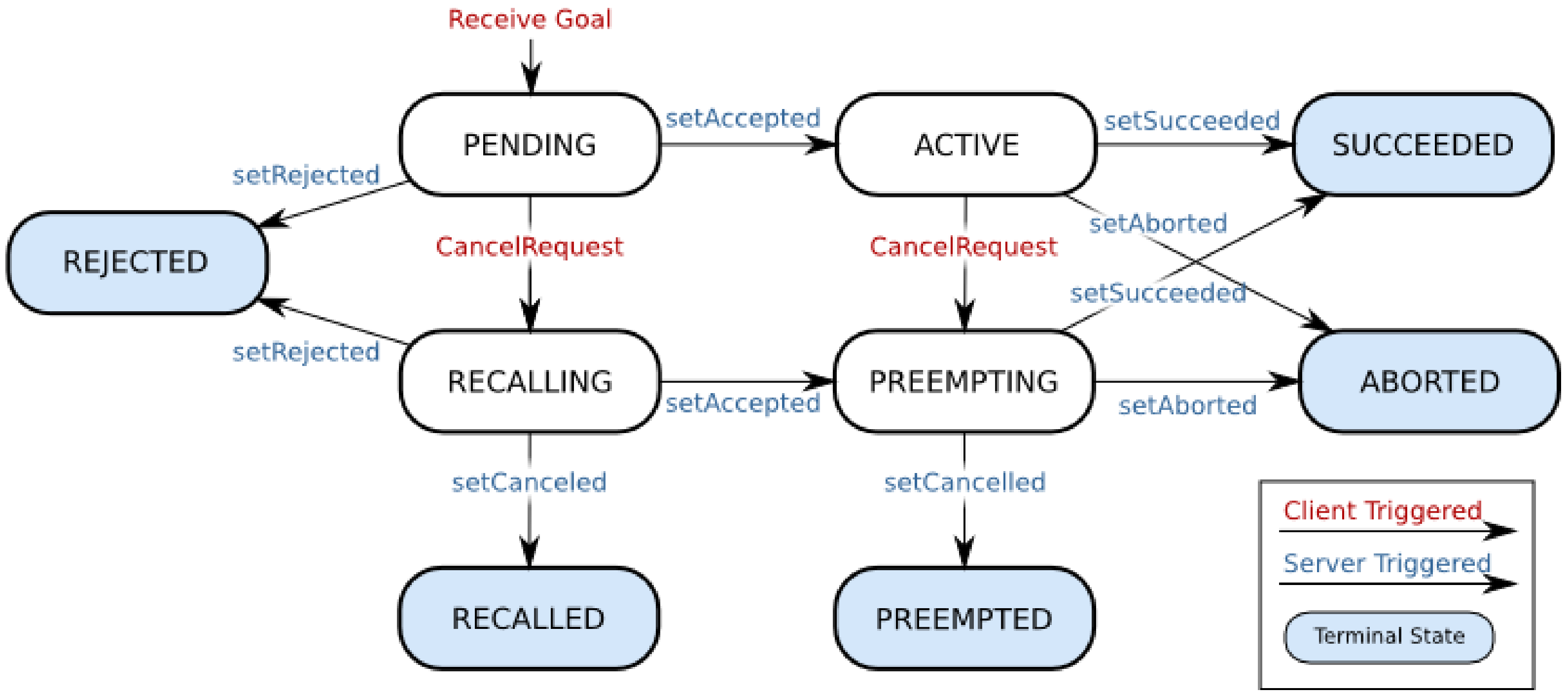
ACTION AND GOAL ID

Action templates are defined by a name and some additional properties through an .action structure defined in ROS

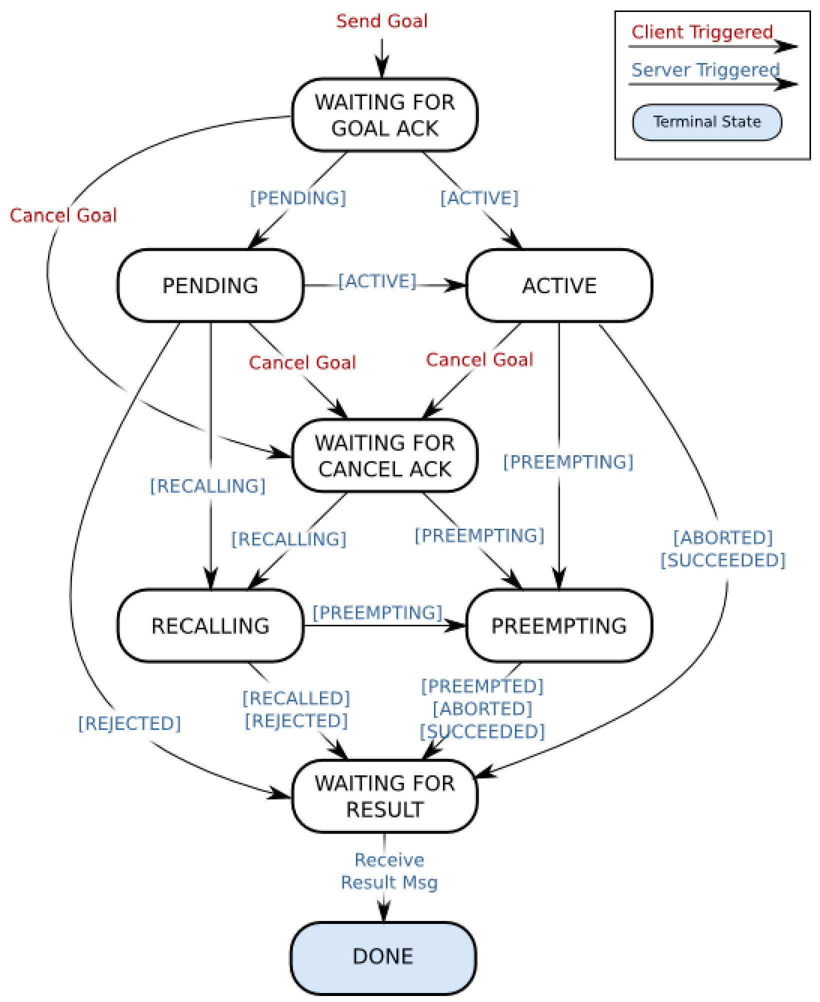
Each ***instance*** of an action has a unique Goal ID

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



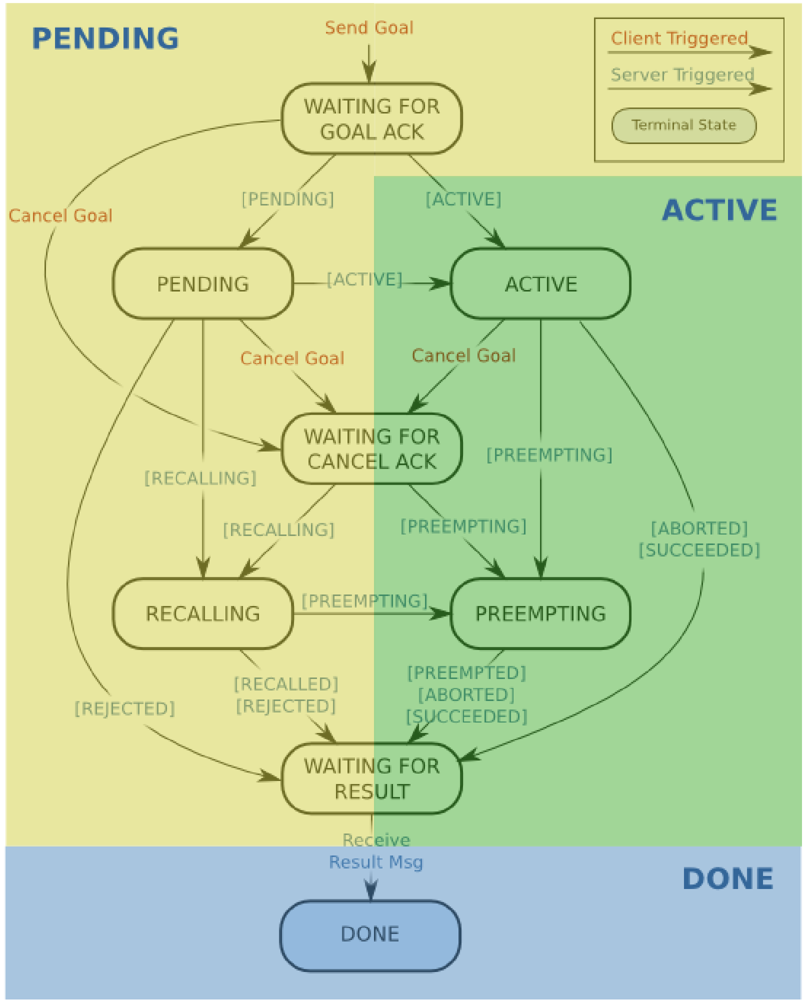
SIMPLEACTIONSERVER/CLIENT

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



.ACTION EXAMPLE

```
# 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
```

SIMPLEACTIONCLIENT

```
#include <chores/DoDishesAction.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<chores::DoDishesAction> Client;
```

SIMPLEACTIONCLIENT

```
int main(int argc, char** argv) {
  ros::init(argc, argv, "do_dishes_client");
  Client client("do_dishes", true); // true -> don't need ros::spin()
  client.waitForServer();
  chores::DoDishesGoal goal;
  goal.dishwasher_id = pickDishwasher();
```

SIMPLEACTIONCLIENT

```
  client.sendGoal(goal);
  client.waitForResult(ros::Duration(5.0));
  if (client.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    ROS_INFO("Yay! The dishes are now clean");
  ROS_INFO("Current State: %s\n", client.getState().toString().c_str());
  return 0;
}
```

USING CALLBACKS

```
client.sendGoal(goal, &doneCb, &activeCb, &feedbackCb);
```

It is possible to add callbacks when providing a goal, to do specific action triggered by certain events

Prototypes:

```
void doneCb(const actionlib::SimpleClientGoalState& state,
            const DoDishesResultConstPtr& result)
void feedbackCb(const DoDishesFeedbackConstPtr& feedback)
void active()
```

SIMPLEACTIONSERVER

```
#include <chores/DoDishesAction.h>
#include <actionlib/server/simple_action_server.h>

typedef actionlib::SimpleActionServer<chores::DoDishesAction> Server;
```

SIMPLEACTIONSERVER

```
void execute(const chores::DoDishesGoalConstPtr& goal, Server* as) {
  while(allClean()) {
    doDishes(goal->dishwasher_id)
    if(as->isPreemptRequested() || !ros::ok()) {
      as->setPreempted();
      break;
    }
    as->publishFeedback(currentWork(goal->dishwasher_id))
  }
  if(currentWork(goal->dishwasher_id) == 100)
    as->setSucceeded();
}
```

SIMPLEACTIONSERVER

```cpp
int main(int argc, char** argv) {
  ros::init(argc, argv, "do_dishes_server");
  ros::NodeHandle n;
  Server server(n, "do_dishes", boost::bind(&execute, _1, &server), false);
  server.start();
  ros::spin();
  return 0;
}
```

COMPILING

Addition in the CMakeList.txt file

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

Addition in the package.xml

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