ROS Bootcamp - Day 3

Introduction To ROS1

JC Cruz and Alec Graves

UTSA RAS

May 17, 2023

Overview

- ROS Message Types
- ROS Services
- ROS Actions (actionlib)
- ROS Time
- ROS Bags
- **Activity:** Custom Package, Autonomous Robot Sim

ROS Message Types: Overview

- ROS messages are used for communication between nodes
- There are built-in message types for common use cases, e.g., sensor data, navigation, and control
- Custom message types can be defined for specific applications
- Custom message types are defined in ’.msg’ files within a package’s ’msg’ folder
- ’.msg’ files contain the data structure of the message

More info: `http://wiki.ros.org/msg`

ROS Message Types: Custom Message Example

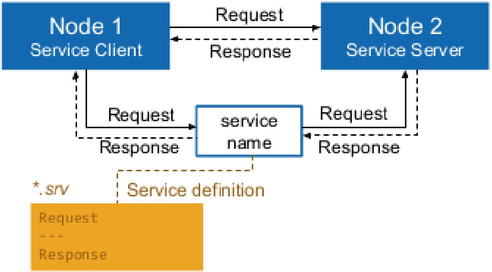
- Create a package with the message generation dependencies: `myrobot_msgs`
- Define the custom message in a '.msg' file

```
string robot_name
float32 x_position
float32 y_position
bool is_active
```

- Modify 'CMakeLists.txt' and 'package.xml' to include message generation
- Build the package to generate the message headers and source files
- Use the custom message type in your publisher and subscriber nodes

ROS Services: Overview

- ROS services are a request-response communication mechanism between nodes
- Consist of a pair of messages: one for the request and one for the response
- Service types are defined in '.srv' files within a package's 'srv' folder
- '.srv' files contain the data structure of the request and response messages



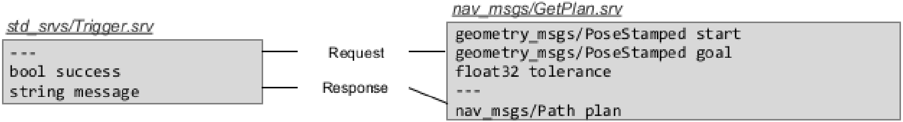
Creating and Using ROS Services

- Define the custom service in a '.srv' file. Place in 'srv' directory.

```
int32 a
int32 b
int32 sum
```

- Modify 'CMakeLists.txt' and 'package.xml' to include service generation
- Build the package to generate the service headers and source files
- Implement the service server and client nodes using the custom service type

ROS Services: Example



std_srvs/Trigger.srv

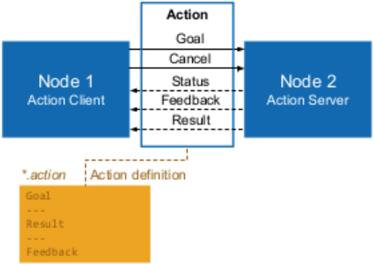
```
---
bool success
string message
```

nav_msgs/GetPlan.srv

```
geometry_msgs/PoseStamped start
geometry_msgs/PoseStamped goal
float32 tolerance
---
nav_msgs/Path plan
```

ROS Actions: Overview

- ROS actions provide asynchronous communication for long-running tasks
- Consist of three messages: goal, feedback, and result
- Action types are defined in ’.action’ files within a package’s ’action’ folder
- ’.action’ files contain the data structure of the goal, feedback, and result messages



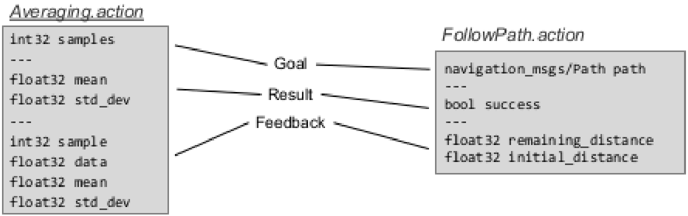
Creating and Using ROS Actions

- Define the custom action in an ’.action’ file

```
int32 goal_number
int32 result_number
int32 current_number
```

- Modify ’CMakeLists.txt’ and ’package.xml’ to include action generation
- Build the package to generate the action headers and source files
- Implement the action server and client nodes using the custom action type

ROS Action: Example



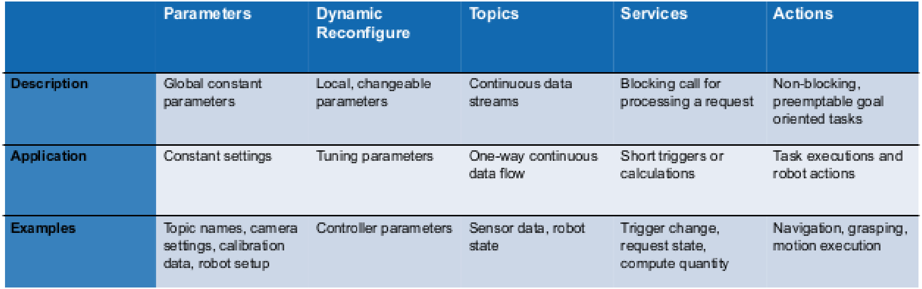
Averaging.action

```
int32 samples
---
float32 mean
float32 std_dev
---
int32 sample
float32 data
float32 mean
float32 std_dev
```

FollowPath.action

```
navigation_msgs/Path path
---
bool success
---
float32 remaining_distance
float32 initial_distance
```

ROS Parameters, Dynamic Reconfigure, Topics, Services, and Actions Comparison



	Parameters	Dynamic Reconfigure	Topics	Services	Actions
Description	Global constant parameters	Local, changeable parameters	Continuous data streams	Blocking call for processing a request	Non-blocking, preemptable goal oriented tasks
Application	Constant settings	Tuning parameters	One-way continuous data flow	Short triggers or calculations	Task executions and robot actions
Examples	Topic names, camera settings, calibration data, robot setup	Controller parameters	Sensor data, robot state	Trigger change, request state, compute quantity	Navigation, grasping, motion execution

ROS Time

- ROS provides a time abstraction that allows you to handle time in a simulation or real-world environment
- Two types of time sources: Wall time and simulation time
- `ros::Time`: Represents time as seconds and nanoseconds since the epoch
- `ros::Duration`: Represents a time span in seconds and nanoseconds
- Use `ros::Time::now()` to get the current time

More info: `http://wiki.ros.org/roscpp/Overview/Time`

ROS Time Example

- Calculate time elapsed between two events:

```
ros::Time start = ros::Time::now();

// Your code here

ros::Time end = ros::Time::now();
ros::Duration elapsed_time = end - start;
ROS_INFO_STREAM("Elapsed time: " <<
elapsed_time.toSec() << " seconds");
```

- Use `ros::Duration` to sleep for a specific duration:

```
ros::Duration sleep_time(1.0); // 1 second
sleep_time.sleep();
```

ROS Bags

- ROS bags are a file format for storing ROS message data
- Useful for recording sensor data, debugging, and playback
- Can record data from multiple topics simultaneously
- Use the `rosbag` command-line tool to record and playback ROS bags

More info: `http://wiki.ros.org/Bags`

Recording ROS Bags

- Record all topics:

```
rosbag record -a -O my_bag.bag
```

- Record specific topics:

```
rosbag record -O my_bag.bag /topic1 /topic2 /
    topic3
```

Playing Back ROS Bags

- Play back a ROS bag:

```
rosbag play my_bag.bag
```

- Play back a ROS bag with specific start and end times:

```
rosbag play my_bag.bag --start=5 --duration=10
```

Activity

Find Today's activity sheet at: \docs\lec3

`https://github.com/UTSARobotics/ros1_bootcamp`