# 08. ROS service, ROS action fogalma, felhasználása

## Elmélet



ZH2 május 26. 10:45 F.05 terem

#### **ROS Service**

- Request/reply communication
- Use a client-server model
- Similar to Remote Procedure Calls (RPC)
- Blocking behavior by default (can be async)

#### **ROS** .srv files

```
int a
int b
---
int sum
```

```
#request constants
int8 FOO=1
int8 BAR=2
#request fields
int8 foobar
another_pkg/AnotherMessage msg
---
#response constants
uint32 SECRET=123456
#response fields
another_pkg/YetAnotherMessage val
CustomMessageDefinedInThisPackage value
uint32 an_integer
```

#### **ROS Action**

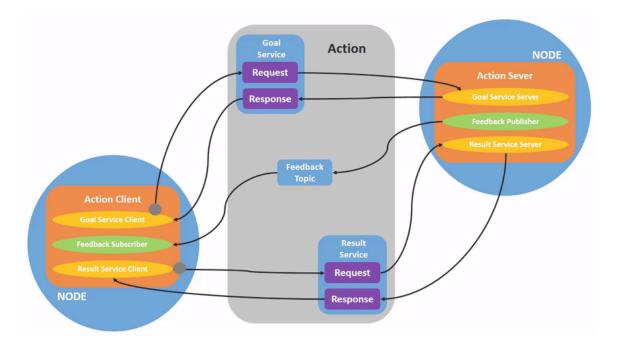


Image source: https://docs.ros.org/

- Ideal for communication with slower processes, e.g. environmental inetarcion
- Asynchronous communication (non-blocking)
- Actions consist of a goal, feedback, and a result
- Built on topics and services and function similarly to services
- Actions are preemptable, (can be canceled while executing)
- Provide steady feedback, unlike services which return a single response
- Actions use a client-server model, similar to the publisher-subscriber model
- An *action client* node sends a goal to an *action server* node that acknowledges the goal and returns a stream of feedback and a result.

#### **ROS** .action files

```
# 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
```

### Building custom interfaces (.msg , .srv and .action files)

- CMakeList.txt and package.xml need to be modified
- For messages and services, see https://docs.ros.org/en/foxy/Tutorials/ Beginner-Client-Libraries/Custom-ROS2-Interfaces.html
- For actions, see https://docs.ros.org/en/foxy/Tutorials/Intermediate/Creatingan-Action.html

# Gyakorlat

#### 1: Actions with Turtlesim

1. Indítsunk el egy turtlesim node -ot és egy turtle\_teleop\_key -t. Figyeljük megy a G| B|V|C|D|E|R|T billentyűk lenyomásának hatását.

```
ros2 run turtlesim turtlesim_node
ros2 run turtlesim turtle_teleop_key
```

2. Tanulmányozzuk a rendszer működését az alábbi parancsok segítségével:

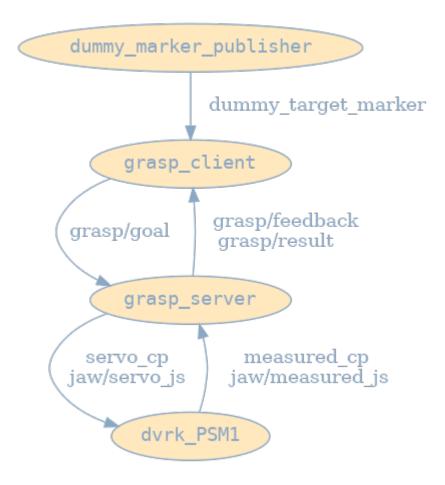
```
ros2 node info /turtlesim
ros2 node info /teleop_turtle
ros2 action list -t
ros2 action info /turtle1/rotate_absolute
ros2 interface show turtlesim/action/RotateAbsolute
```

3. Küldjünk action goal-t a parancssorból:

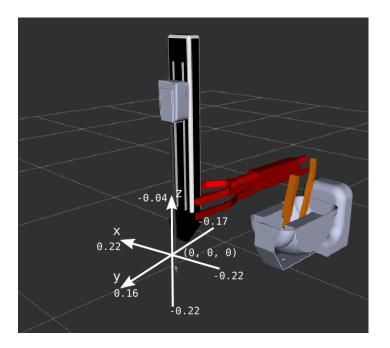
 $ros2\ action\ send\_goal\ / turtle 1/rotate\_absolute\ turtle sim/action/Rotate Absolute\ "\{theta: 3.14\}"$ 

### 2: PSM grasp action

A következőkben ROS action server--client architektúrát fogunk implementálni. A korábban implementált psm\_grasp.py funkcionalitását fogjuk két külön node-ra bontani az alább ábra szerint.



#### 2.1: dVRK ROS2



1. Indítsuk el a PSM1 RViz szimulációját. A dVRK konzolon ne felejtsünk el HOME-olni.

# ROS 2 joint and robot state publishers ros2 launch dvrk\_model dvrk\_state\_publisher.launch.py arm:=PSM1

# RViz ros2 run rviz2 rviz2 -d ~/ros2\_ws/install/dvrk\_model/share/dvrk\_model/rviz/PSM1.rviz

# URDF-fel kapcsolatos hibák esetén

sudo apt update && sudo apt install locales sudo locale-gen en\_US en\_US.UTF-8 sudo update-locale LC\_ALL=en\_US.UTF-8 LANG=en\_US.UTF-8 export LANG=en\_US.UTF-8

locale # verify settings

locale # check for UTF-8

2. Tanulmányozzuk a szimulátor működését ROS a tanult prancsok (ros2 topic list, ros2 topic echo ros2 run rqt gui rqt gui, stb.) használatával.

#### 2.2: Grasp action létrehozása

1. Hozzunk létre új csomagot ros2\_course\_msgs névvel:

```
cd ros2_ws/src
ros2 pkg create --build-type ament_cmake ros2_course_msgs --dependencies
action_msgs std_msgs geometry_msgs rosidl_default_generators
```

2. Hozzuk létre a ros2 course msgs/action/Grasp.action fájlt az alábbi tartalommal:

```
# Goal
geometry_msgs/Point grasp_pos
---
# Result
bool success
---
# Feedback
string status
```

3. Adjuk hozzá a következőt a CMakeLists.txt-hez az ament\_package() sor elé:

```
rosidl_generate_interfaces(${PROJECT_NAME}
   "action/Grasp.action"
   DEPENDENCIES geometry_msgs
)
```

4. Adjuk hozzá a dependency-ket a package.xml -hez:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

5. Build-eljük a workspace-t:

```
cd ~/ros2_ws
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
```

6. Ellenőrizzük, hogy létrejött-e a Grasp action:

```
source ~/ros2_ws/install/setup.bash
ros2 interface show ros2 course msgs/action/Grasp
```

7. A ros2\_course csomag package.xml -jéhez adjuk hozzá a következő sort a ros2 course msgs dependency beállításához:

```
<exec depend>ros2 course msgs</exec depend>
```

#### 2.2: Grasp server implementálása

- 1. Hozzunk létre új python forrásfájlt grasp\_server.py névvel a ~/ros2\_ws/src/ros2\_course/ros2\_course mappában. Adjuk meg az új entry point-ot a setup.py-ban a megszokott módon.
- 2. Implementáljuk az action server-t. Induljunk ki a példából: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Writing-an-Action-Server-Client/Py.html. A node funkcionalitása átemelhető a psm\_grasp.py-ból. A node egy Grasp action-t vár, melynek hatására megragadja a goal-ban szerepló koordinátákon található tárgyat (a dummy markert).

#### **CLion interpreter**

CLion-ban az interpreterhez adjuk hozzá a /home/tamas/ros2\_ws/install/ros2 course msgs/lib/python3.8/site-packages elérési utat.

3. Futtassuk az action server-t és teszteljük a működését parancssorból:

```
ros2 run ros2 course grasp server
```

```
ros2 node info grasp_server
ros2 action list -t
ros2 action info /grasp
ros2 interface show ros2_course_msgs/action/Grasp
ros2 action send_goal --feedback /grasp ros2_course_msgs/action/Grasp "{grasp_pos:
{x: 0.0, y: 0.0, z: -0.18}}"
```

#### 2.3: Grasp client implementálása

- 1. Hozzunk létre új python forrásfájlt grasp\_client.py névvel a ~/ros2\_ws/src/ros2\_course/ros2\_course mappában. Adjuk meg az új entry point-ot a setup.py-ban a megszokott módon.
- 2. Implementáljuk az action client-et. Induljunk ki a példából: https://docs.ros.org/en/foxy/Tutorials/Intermediate/Writing-an-Action-Server-Client/Py.html. A node funkcionalitása átemelhető a psm\_grasp.py-ból. A node feliratkozik a /dummy\_target\_marker topic-ra, és egy Grasp action request-et küld az action server-nek (`grasp\_server') a marker megragadására.
- 3. Futtassuk az action client-et (grasp client) és a dummy marker-t:

ros2 run ros2 course dummy marker

ros2 run ros2\_course grasp\_client

#### 3. IsPrime service

1. Implementáljunk ROS Service-t, amely egy kapott természetes számról megállapítja, prím-e. Használjuk az implementált service-t. Kiindulhatunk ebből a példából: https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Service-And-Client.html.

#### Hasznos linkek

- Build dVRK2 on ROS2
- ROS 2 examples
- Understanding ROS actions
- · Creating an action
- Writing an Action Server and Client

• https://docs.ros.org/en/foxy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Service-And-Client.html