UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Comunicación entre Nodos en ROS2: Creación y Manipulación de Servicios y Clientes

Eugenio Ivorra

En esta práctica, se busca desarrollar una comprensión teórica y práctica de la comunicación entre nodos en ROS2 mediante la creación y manipulación de servicios y clientes en Python, y aplicar este conocimiento en un contexto práctico usando el simulador turtlesim:

- Comprender la estructura y funcionamiento de los servicios en ROS2, y cómo se diferencian del modelo de publicador-suscriptor.
- Crear un nodo servidor de servicio en Python que pueda recibir una solicitud, procesarla y enviar una respuesta, utilizando el ejemplo de sumar dos enteros.
- Crear un nodo cliente en Python que pueda enviar una solicitud a un servicio y procesar la respuesta recibida.
- Integrar y aplicar los conceptos aprendidos sobre servicios en ROS2 mediante el desarrollo de una aplicación práctica que controle el movimiento de una tortuga en el simulador turtlesim, utilizando servicios para enviar comandos de posición.

Índice general

1 Programación de Servicios

En esta práctica, aprenderemos cómo crear un servicio y un cliente en ROS2 usando Python. Para aprender la estructura de los servicios revisaremos los paquetes de ejemplo y en concreto el del servicio simple que suma dos enteros.

1.1 Introducción

Los servicios constituyen otra modalidad de comunicación entre los nodos dentro del grafo de ROS2. A diferencia de los topics, que operan bajo un modelo de publicador-suscriptor, los servicios se rigen por un modelo de llamada y respuesta.

En el modelo de publicador-suscriptor, los nodos se suscriben a tópicos específicos y reciben actualizaciones continuas cada vez que hay nueva información disponible en esos tópicos. Este modelo es especialmente útil en situaciones donde los datos se generan o cambian frecuentemente, y los nodos requieren estar al tanto de estos cambios en tiempo real.

Por otro lado, el modelo de llamada y respuesta de los servicios es más interactivo y controlado. En este modelo, un nodo cliente realiza una solicitud a un nodo servidor a través de un servicio, y luego espera una respuesta. El nodo servidor procesa la solicitud y envía una respuesta de vuelta al nodo cliente. Los servicios son particularmente útiles en situaciones donde los nodos necesitan realizar peticiones específicas y obtener respuestas concretas, en lugar de recibir transmisiones de datos continuas.

Este modelo de operación permite una comunicación más controlada y dirigida, lo que puede ser crucial en operaciones que requieren precisión y una interacción detallada entre los nodos. Por lo tanto, mientras que los topics son excelentes para la distribución amplia y continua de datos, los servicios proporcionan un mecanismo para la interacción punto a punto basada en demanda entre los nodos en la red de ROS.

En la siguiente figura 1.1 se muestra un gráfico de la comunicación por servicios entre dos nodos. Es posible que existan varios nodos clientes que realicen peticiones simultáneas sobre el mismo servicio pero solo puede haber un nodo servidor ofreciendo dicho servicio.

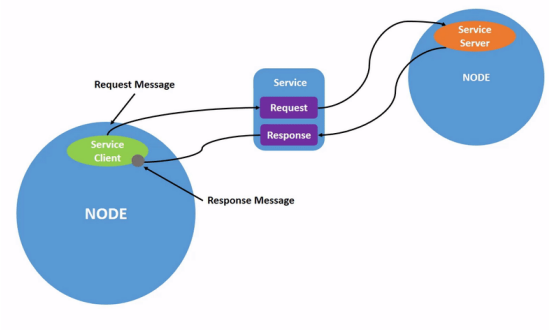

Figura 1.1: Ejemplo gráfico de una comunicación por servicios

1.2 Examinar el Código del Servidor

Navega y abre el fichero `~/ros2_ws/src/ROS2_examples/rclpy/services/minimal_service/examples_rclpy_minimal_service/service_member_function.py` con tu editor de texto favorito.

Vamos a comenzar con el servidor del servicio. Primero, se importa el tipo de servicio `AddTwoInts` del paquete `example_interfaces`. A continuación, se importa la biblioteca de cliente Python de ROS 2 y, específicamente, la clase `Node`.

```
from example_interfaces.srv import AddTwoInts
import rclpy
from rclpy.node import Node
```

La clase `MinimalService` inicializa el nodo con el nombre `minimal_service`. Luego, crea un servicio y define su tipo, nombre y callback.

```
def __init__(self):
    super().__init__('minimal_service')
    self.srv = self.create_service(AddTwoInts, 'add_two_ints',
    ↪  self.add_two_ints_callback)
```

La definición del callback del servicio recibe los datos de la solicitud, los suma y devuelve la suma como respuesta.

```
def add_two_ints_callback(self, request, response):
    response.sum = request.a + request.b
    self.get_logger().info('Incoming request\na: %d b: %d' % (request.a,
    ↪  request.b))
    return response
```

Finalmente, el método main inicia los servicios de ROS instancia la clase `MinimalService` para crear el nodo del servicio y mantiene el nodo en ejecución para manejar los callbacks.

```
def main(args=None):
    rclpy.init(args=args)

    minimal_service = MinimalService()

    rclpy.spin(minimal_service)

    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

1.3 Punto de Entrada Servidor

Para permitir que el comando `ros2 run` ejecute tu nodo servidor , tiene que haber un punto de entrada al archivo `setup.py` igual que hacíamos con los publicadores y suscriptores.

Comprueba que aparece la siguiente línea entre los corchetes de `'console_scripts'`:

```
'service_member_function = '
            ' examples_rclpy_minimal_service.service_member_function:main',
```

1.4 Probar el servidor

Ya sabemos cómo llamar a un servicio usando la herramienta `rqt` pero estas llamadas también se pueden realizar desde la terminal con esta estructura:

```
    ros2 service call <service_name> <service_type> <arguments>
```

Para probar nuestro servidor de ejemplo primero lanzaríamos el nodo servidor:

```
    ros2 run examples_rclpy_minimal_service service_member_function
```

Podemos comprobar que el servicio está activo con el comando:

```
    ros2 service list -t
```

Donde nos tiene que aparecer el nombre del servicio y el tipo de mensaje que espera.

```
/add_two_ints [example_interfaces/srv/AddTwoInts]
```

Con la herramienta de línea de comandos `ros2 interface show` podemos explorar la interfaz de un servicio o un topic específico. A continuación se muestra cómo utilizar este comando para examinar el servicio `/add_two_ints` que utiliza la interfaz `example_interfaces/srv/AddTwoInts`.

Abre una terminal y ejecuta el siguiente comando:

```
ros2 interface show example_interfaces/srv/AddTwoInts
```

El comando anterior mostrará la definición de la interfaz de servicio en la terminal. La salida debería ser algo similar a lo siguiente:

```
int64 a
int64 b
---
int64 sum
```

En la definición de la interfaz mostrada, se observa que el servicio `/add_two_ints` toma dos enteros `a` y `b` como entrada y devuelve la suma de estos dos números como `sum`. Esto se indica por la línea de separación `---` que divide los campos de entrada y salida del servicio.

Y por último haríamos la llamada donde estamos pasándole como argumentos los números 2 y 5 en otra terminal.

```
    ros2 service call /add_two_ints example_interfaces/srv/AddTwoInts "{a: 2
    ↪  , b: 5}"
```

1.5 Examinar el código del Nodo Cliente

Dentro del directorio ~/ros2_ws/src/ROS2_examples/rclpy/services/minimal_client/examples_rclpy_minimal_client/ se encuentra el archivo llamado `client_async_member_function.py`

La única importación diferente para el cliente es `import sys`. El código del nodo cliente usa `sys.argv` para acceder a los argumentos de entrada de la línea de comandos para la solicitud.

```
def main():
    rclpy.init()

    minimal_client = MinimalClientAsync()
    response = minimal_client.send_request(int(sys.argv[1]),
    ↪  int(sys.argv[2]))
    minimal_client.get_logger().info(
```

```
        'Result of add_two_ints: for %d + %d = %d' %
        (int(sys.argv[1]), int(sys.argv[2]), response.sum))

    minimal_client.destroy_node()
    rclpy.shutdown()
```

La definición del constructor crea un nodo y luego el cliente con el mismo tipo y nombre que el nodo del servicio. El tipo y el nombre deben coincidir para que el cliente y el servicio puedan comunicarse. El bucle `while` verifica si un servicio que coincide con el tipo y el nombre del cliente está disponible una vez por segundo. Después crea el mensaje de petición del tipo adecuado.

```
    def __init__(self):
        super().__init__('minimal_client_async')
        self.cli = self.create_client(AddTwoInts, 'add_two_ints')
        while not self.cli.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('service not available, waiting again...')
        self.req = AddTwoInts.Request()
```

Por último se implementa el método que realiza la petición al servicio rellenando la variable de `request` y esperando hasta recibir la respuesta del servidor.

```
    def send_request(self, a, b):
        self.req.a = a
        self.req.b = b
        self.future = self.cli.call_async(self.req)
        rclpy.spin_until_future_complete(self, self.future)
        return self.future.result()
```

1.6 Punto de Entrada Cliente

Al igual que con el servidor, también tiene que haber un punto de entrada para poder ejecutar el nodo cliente.

El campo `entry_points` de tu archivo `setup.py` debería tener esta línea:

```
    entry_points={
        'console_scripts': [
            'client_async_member_function ='
            '
            ↪  examples_rclpy_minimal_client.client_async_member_function:main',
        ],
```

1.7 Prueba del nodo cliente y servidor

Ahora ejecuta el nodo del servidor como hemos visto anteriormente y en otra terminal el nodo cliente con dos enteros separados por un espacio:

```
ros2 run examples_rclpy_minimal_client client_async_member_function 2 3
```

Si elegiste 2 y 3, por ejemplo, el cliente recibiría una respuesta como esta:

```
[INFO] [minimal_client_async]: Result of add_two_ints: for 2 + 3 = 5
```

Regresa a la terminal donde se está ejecutando tu nodo de servicio. Verás que publicó mensajes de log cuando recibió la solicitud:

```
[INFO] [minimal_service]: Incoming request
a: 2 b: 3
```

Presiona Ctrl+C en la terminal del servidor para detener el nodo.

2 Actividad

Desarrolla un nodo servidor para gestionar el movimiento de la tortuga en `turtlesim`. El servicio deberá denominarse `moverTortuga`, y deberá recibir una cadena de texto (String) como entrada. Esta cadena de texto especificará la figura que la tortuga debe dibujar: un cuadrado o un círculo. Si la cadena de texto recibida no corresponde a ninguna de estas figuras, la tortuga no debe realizar ninguna acción. Asegúrate de mostrar por consola el movimiento solicitado cada vez que se reciba una nueva solicitud y devolver en la respuesta una String con el movimiento ejecutado. La interfaz a usar es `StringService` del paquete `minimal_interfaces`.

```
ros2 interface show minimal_interfaces/srv/StringService
```

Si el comando anterior te da error de que no encuentra la interfaz, actualiza el repositorio de `ROS2_examples`, vuelve a compilar y recarga el source.

Para dibujar las figuras, puedes utilizar la misma lógica de movimiento de la tortuga que has desarrollado previamente con los publicadores y subscriptores.

Una vez que hayas implementado y probado el servicio, crea un nodo cliente que realice tres peticiones al servidor con el objetivo de dibujar un cuadrado, un círculo y luego otro cuadrado. Cada una de estas peticiones debe ser procesada por el servidor, que guiará a la tortuga para dibujar la figura correspondiente en la simulación de `turtlesim`.

Bibliografía

Web de los tutoriales de ROS (2023). https://docs.ros.org/en/humble/Tutorials.html.

Web de ROS2 Humble (2023). https://docs.ros.org/en/humble/index.html.