Overview

This document introduces you to the Robot Operating System (ROS) and its most popular communication style: topics. The following lecture will discuss quality attributes and trade-offs resulting from the design decisions that were made by the ROS framework developers and how they impact systems that use topics as communication style. After reading this, try to answer the following questions, which we will discuss in the up-coming lecture:

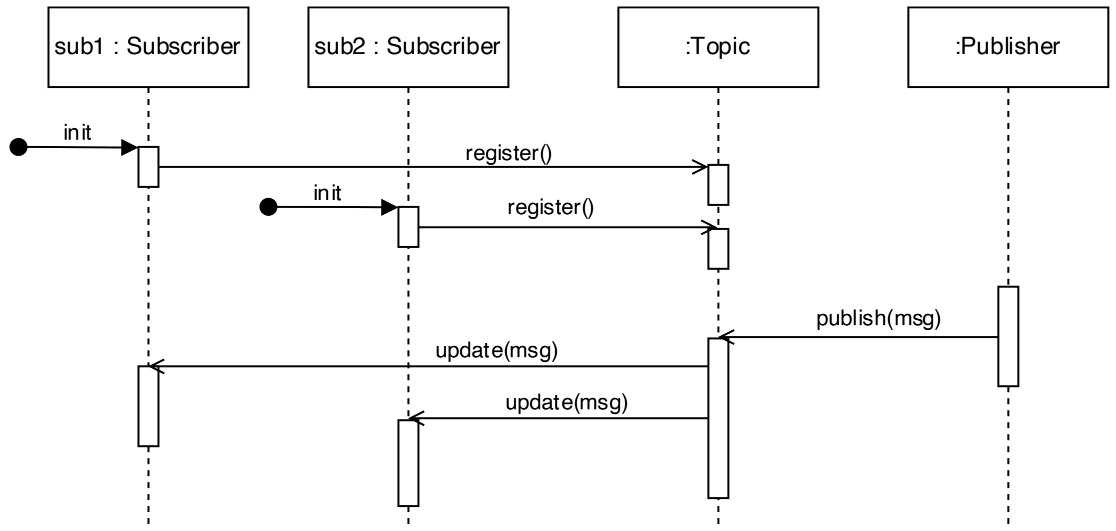
- **What are advantages and disadvantages of using publish-subscribe connectors over sending messages to components directly?**
- **What are advantages and disadvantages of the way how ROS implements publish subscribe?**
- **What could potentially go wrong when complex systems that use ROS Topics evolve over many years?**

What is ROS?

ROS is the most popular open-source framework for component-based robotics systems. ROS has been deployed to a diversity of robots, including autonomous vehicles, mobile manipulators, underwater systems, and humanoids in environments ranging from industrial warehouses to the International Space Station. To increase the reusability of the more than 7400 software packages in the ROS ecosystem, ROS uses configuration mechanisms and connectors that are not bound during compile time. These mechanisms include string-based identifiers for topics, services, actions, and parameters, as well as re-mappings between these identifiers.

What is Publish-Subscribe?

Publish-subscribe is a domain-independent component communication pattern for asynchronous message sending that loosely couples senders (i.e., *publishers*) from receivers (i.e., *subscribers*) via a known intermediary interface (i.e., *publish-subscribe* connector) that functions as a layer of indirection. Similar to the Observer design pattern, after subscribing to a topic, a connector is added between the subscriber and all publishers of the corresponding message type and the subscriber starts to receive the messages whenever any corresponding publisher sends them. Depending on the implementation or configuration, subscribers also receive all messages previously published at the message type. Unsubscribing removes the connector to the corresponding subscriber.



How does Publish-Subscribe Differ from Direct Communication?

In direction communication the sender explicitly specifies the receiver(s) of messages rather than sending it to a topic to which receivers can subscribe themselves. Therefore, direct communication assumes that the sender of messages knows which other components need the data that the sender produces.

How to deal with Subscribers Receiving Multiple Messages?

Since publish-subscribe is intended for asynchronously communicating components, messages might be received while the component is still processing another message. For this purpose, publish subscribe infrastructures provide **message queues** at the input port of each component so that messages can wait until the component is ready to process them. Messages queues usually have a fixed maximum size.

Where is Publish-Subscribe used?

Publish-Subscribe is often used in component-based systems that process discrete streams of data (e.g., cyber-physical systems, IoT systems) or event-driven architectures (e.g., microservice systems, enterprise systems, logging and monitoring systems). Besides ROS, there are many other frameworks and library that implement publish-subscribe, such as MQTT, or RabbitMQ, Amazon Web Services, Kafka, …

What are ROS Topics?

Topics implement a publish-subscribe style providing asynchronous message-based, multi-endpoint communication between nodes. Components subscribe to topics using the string-representation of their name and name space. Then they receive any data published to the subscribed topics. There can be multiple publishers and subscribers for a topic. Topics are the main form of communication between nodes in ROS, and are used for periodic information (e.g., sensor data or positions) or sporadic requests, such as turning off a motor.

How are ROS Topics implemented?

Publishers are created with the advertise API call and include the topic name as a string:

```
ros::Publisher pub = nh.advertise<std_msgs::String>("topic_name", 5);
std_msgs::String str;
str.data = "hello world";
pub.publish(str);
```

Subscribers subscribe to topics with the subscribe API call, using the corresponding topic name as a string:

```
void callback(const std_msgs::StringConstPtr& str)
{
  ...
}
  ...
ros::Subscriber sub = nh.subscribe("topic_name", 1, callback);
```

You can read more about the APIs in the ROS Wiki:
http://wiki.ros.org/roscpp/Overview/Publishers and Subscribers