eProsima

The Middleware Experts

ROS2 & DDS

ROS2

Borja Outerelo Gamarra
Senior Software Developer
borjaouterelo@eprosima.com

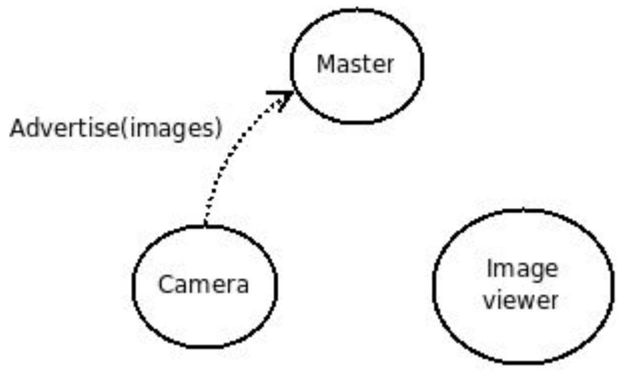
www.eprosima.com

Agenda

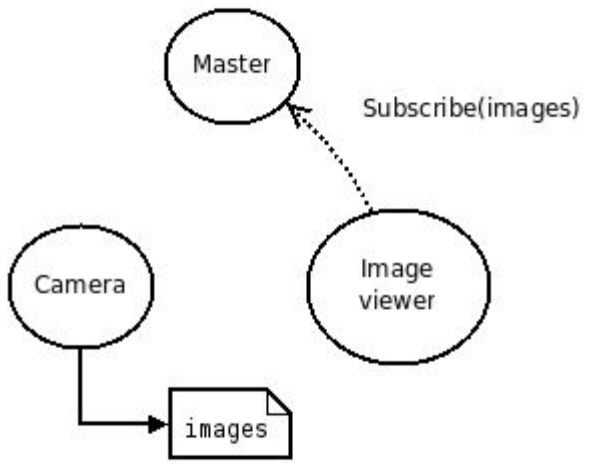
- ROS Concepts.
- From ROS to ROS2.
- DDS in ROS2.
- ROS2 Concepts.
- Example.

ROS Concepts

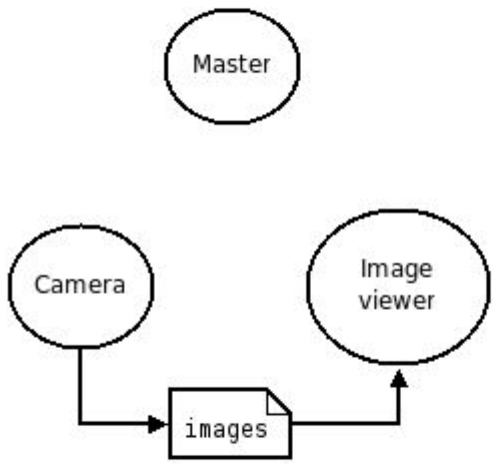
- ROS master:
  - Centralized piece for ROS communications.
  - Allows Nodes discovery.
  - Register ROS Computation Graph. (peer-to-peer network of loosely coupled processes)



1



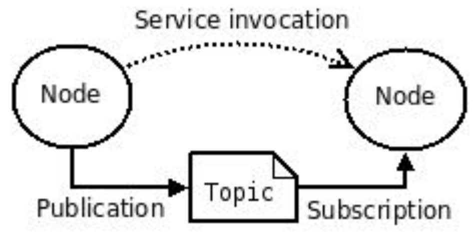
2



3

ROS Concepts

- Nodes:
  - Process controlling a part of a Robot. (Laser, motors, planner …)
  - Communicates between them.
  - Developed using ROS client library.



- Parameter Server
  - Centralized in the ROS master.
  - Uses key-value representation.
- Messages
  - Unit of communication.
  - .msg files.

ROS Concepts

- Topics
  - Stream of messages.
  - Strongly typed by message type.
  - Central part of communications: publish/subscribe.
  - Transports: TCPROS/UDPROS.
- Services
  - Request/Reply communication model Service <-> Client.
  - RPC.
  - .srv files.
- Bags
  - Stores messages from a topic.
  - Playback mechanism.

From ROS to ROS2

- New use cases: autonomous vehicles, multi-robot swarms and operating in distributed environments...
- Initially designed for single robot control with no real-time requirements and using reliable connections.
- ROS uses a centralised discovery. (Single point of failure) ROS 2 is fully distributed including discovery.
- Is not just an API change it changed the underlying communications.

DDS in ROS2

- No reinvent the wheel!
  - DDS implementations usually do not introduce dependencies.
  - DDS is end-to-end vs build from multiple software -> dependencies
  - Documented, formal specification and API.
    - DDS API in OMG specification -> ROS2 is DDS vendor independent.
      - Provide power users a choice point. ROS2 defaulted FastRTPS.

- DDS match ROS requirements
  - Discovery.
  - Message definition and serialization.
  - Publish-subscribe transport.
  - DDS-RPC.

DDS in ROS2

- ROS-like interface:
  - DDS concepts: participant, subscriber, publisher.
  - Topic-oriented.

- DDS discovery:
  - Distributed discovery system.
  - No need for a centralized system: ~~ROS master~~.
    - +fault tolerant +flexible.
    - Options for static discovery.

- DDS Quality of Service (QoS).
  - Flexible.
  - Communication control. Improved on some networks.

DDS in ROS2

- DDS IDL
  - ROS 2.0 API works with the ROS .msg and .srv.
  - Copy from .msg, .srv in memory representation to DDS messages.
    - Serialisation step is worst.

- Services
  - ROS 2 implementation on top of publish/subscribe DDS API + reliable QoS.
  - DDS RPC -> OMG Standard.

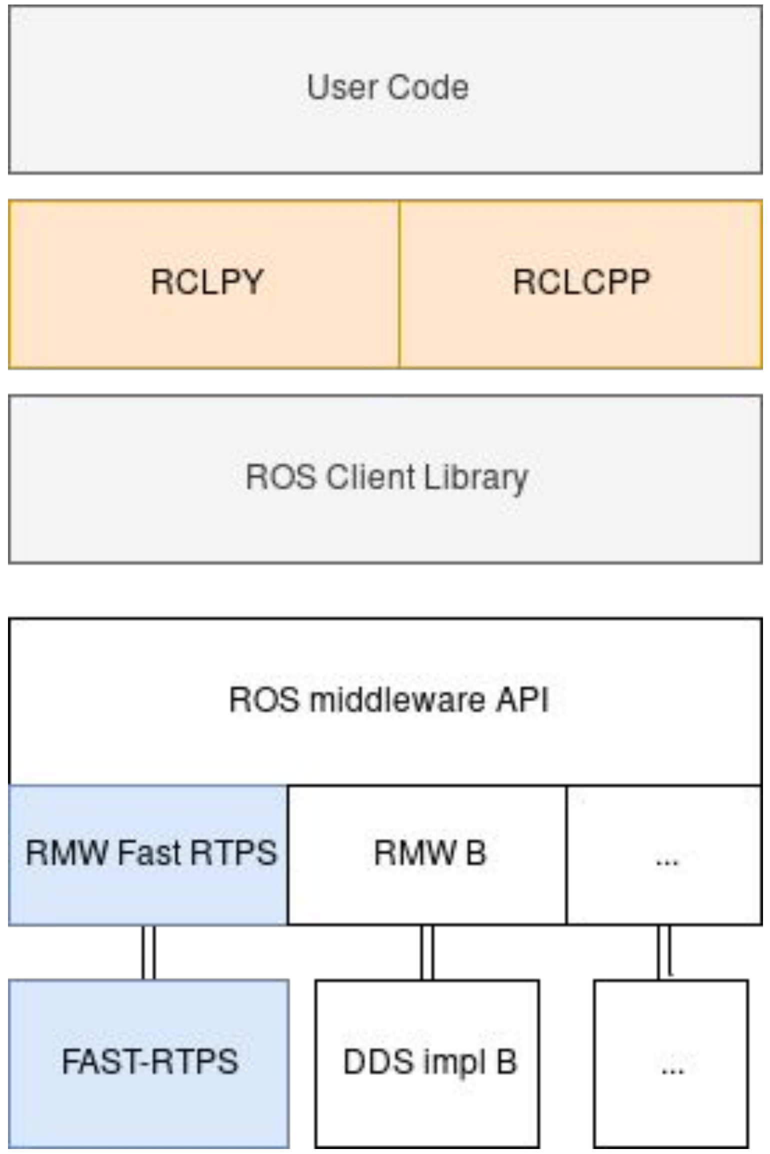
DDS in ROS2

- Allows support of multiple languages:
  - C DDS API -> bindings.
  - C++ DDS API -> wrapper + bindings.

- DDSI-RTPS (DDS-Interoperability Real-Time Publish-Subscribe) protocol replace ROS’s TCPROS and UDPROS wire protocols for publishing/subscribe.

RO2 Concepts

- Architecture:



RO2 Concepts

- Same as in ROS:
  - Nodes.
  - Services.
  - Publishers and subscribers.
  - Messages.
  - Topics.

Demo

1. Install ROS2: https://index.ros.org/doc/ros2/Installation/.
2. Install demo packages.
3. source /opt/ros/crystal/setup.bash
   - This will set the environment with multiple tools.
     - ros2 pkg list
     - ros2 node list
     - ros2 topic list
   - In another terminal, launch a node:
     - ros2 run demo_nodes_cpp talker
     - Now you can play with "ros2 node" and "ros2 topic" commands.

Demo

Let's create our first package:

- Colcon is a helper build-tool:
  a. apt install python3-colcon-common-extensions
- ros2 pkg create command:
  a. --cpp-node-name
  b. --dependencies rclcpp std_msgs

Demo

Let’s publish something:

```
#include "rclcpp/rclcpp.hpp"

class MinimalPublisher : public rclcpp::Node
{
public:
  MinimalPublisher()
  : Node("minimal_publisher"), count_(0)
  {

}
private:
      size_t count_;
};
```

Demo

Let’s publish something:

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

class MinimalPublisher : public rclcpp::Node
{
public:
  MinimalPublisher()
  : Node("minimal_publisher"), count_(0)
  {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic");

}
private:
      rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
      size_t count_;
};
```

Demo

Let’s publish something:

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"
using namespace std::chrono_literals;
class MinimalPublisher : public rclcpp::Node
{
public:
  MinimalPublisher()
  : Node("minimal_publisher"), count_(0)
  {
      publisher_ = this->create_publisher<std_msgs::msg::String>("topic");
      auto timer_callback =
            [this]() -> void {
                  auto message = std_msgs::msg::String();
                  message.data = "Hello, world! " + std::to_string(this->count_++);
                  RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
                  this->publisher_->publish(message);
            };
      timer_ = this->create_wall_timer(500ms, timer_callback);
}
private:
      rclcpp::TimerBase::SharedPtr timer_;
      rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
      size_t count_;
};
```

Demo

Let’s publish something:

```
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

Demo

Let's create our first package:

- Build using colcon:
  - colcon build --packages-select <package-name>
- Source environment from install directory.
- ros2 run <package-name> <node-name>

- We can check it works in a different terminal with:
  - ros2 topic echo topic

References

This presentation has been created from the following original content:

- ROS2 design: https://design.ros2.org/
- ROS2 demos: https://index.ros.org
- Examples: https://github.com/ros2/examples

eProsima
The Middleware Experts

Thank you!

Borja Outerelo Gamarra
Senior Software Developer
borjaouterelo@eprosima.com

www.eprosima.com