The ROS 2 Vision

For Advancing the Future
of Robotics Development

Sep. 21st 2017
Dirk Thomas, Mikael Arguedas
ROSCon 2017, Vancouver, Canada
"Unboxing"
Getting Started

command data

video data

open robotics
Moving Outdoor

- command data
- video data
Quality of Service

Computer

joystick node

/cmd_vel
reliable

flight cntrl node

Robot

image view node

/image
best effort

camera node
Extend Capabilities: Hardware

Computer
- image view node
- image view node #2

Robot
- camera node
- camera node #2

Renaming topic at runtime

Image view node -> /image
/Image2 -> /image2
Extend Capabilities: Software

Robot

/command_velocity

/flight_controller_node

/obstacles

/left_camera_node

/stereo_image_pipeline_node

/right_camera_node

/image_left

/image_right
Recap #1

- Multi-OS support: Linux, Mac OS, Windows
  - Binary packages for Mac OS and Windows

- Different client libraries share common implementation

- Quality of Service: variety of configuration options
  - DDS provides even more configuration options, directly accessible

- Hardware with "native" communication interface
  (no need for separate protocols and driver packages)

- Event based notifications (rather than need for polling)
  - Remapping of topics at runtime
Undeterministic Startup

Robot

nodes ready

flight cntrl node
/cmd_vel

no obstacles reported

/obstacles

left camera node
/image_left

node not ready

right camera node
/image_right

stereo image pipeline node
Lifecycle State Machine

unconfigured → inactive → finalized

inactive ← active ← inactive
Deterministic Startup

Robot

all active

/cmd_vel

flight cntrl
node

/obstacles

stereo image
pipeline node

left camera
node

left camera
node

right camera
node

Ready to fly

/image_left

/image_right
Coexistence with ROS 1

- `/cmd_vel` (reliable)
- `/image` (best effort)

Running ROS 1
Usage Patterns of the "ros1_bridge"

Beta 2
- TurtleBot 2 demo
- Robot using ROS 2 onboard
- Computer uses ROS 1 tools, leverage existing ROS 1 packages

Beta 3
- HSR demo (see ⚡ talk)
- Robot using ROS 1 onboard
- Computer uses ROS 2 tools, leverage intrinsic advantages of the communication protocol
Usage of the "ros1_bridge"

- `/cmd_vel` (reliable)
- `/image` (best effort)
Usage Patterns of the "ros1_bridge"

Beta 2
- TurtleBot 2 demo
- Robot using ROS 2 onboard
- Computer uses ROS 1 tools, leverage existing ROS 1 packages

Beta 3
- HSR demo (see ⚡ talk)
- Robot using ROS 1 onboard
- Computer uses ROS 2 tools, leverage intrinsic advantages of the communication protocol

- Incrementally migrate a ROS system
Multi Robot

- Distributed discovery useful for on-demand robot-to-robot comm.
  - All current ROS middleware implementations support it
  - As long as all robots use the same comm. protocol
    they can communicate (independent of the vendor)

- Quality of Service settings to tailor the comm. for the specific scenario
  (see eProsima's talk @ 11:35)

- Dynamic remapping of topics enables various different approaches, e.g.:
  - Flip namespaced robot spec. topics to be "global"
    `/robotA/pose` → `/pose`
  - Subscribe to a specific topic from a group of robots
    `/**/pose` or `/floor2/*/pose`
Adding a Custom Sensor

- **Robot**
  - MCU module
  - Bluetooth receiver node
  - /found_beacon
  - GPS receiver node
  - Pose estim. node

- **Home**
  - Beacon
  - Same ROS 2 code!
Flight Controller Internals

Flight controller node

Subscription thread

Realtime thread

Publication thread

/found_beacon

/return_home
Recap #2

- Lifecycle nodes
- Basic Python-based launch files
  - Launch utilizing lifecycle state machine
- Dual-home bridge to exchange msgs / srvs between ROS 1 and ROS 2
  - More configuration options
- Multi robot benefiting from the communication protocol:
  - distributed discovery, configurable QoS, dynamic remapping
- "Native" communication protocol with micro controllers (DDS-XRCE)
- Proof of concept for real time support using custom allocators
  - No usage of real time kernel yet, no continuous testing
Process Layout Decision

Robot

Save configuration as launch file

Single Process

flight cntrl node

left camera node

right camera node

stereo image pipeline node

/cmd_vel

/image_left

/image_right
Fault Tolerance and Fallback Behaviors

Fallback behavior #2: switch to monocular det.

right camera node ➔ /image ➔ mono. obst. detect node
Unsecure System

command data

Command spoofed

video data
Securing the System

- Command data
- Authentication
- Encrypt sensitive data
- Video data

openrobotics
Validation and Certification

- Use certified hardware components (which talks DDS)
- Use certified DDS implementation
- Use certified / validated software components

- Select only the subsystems of ROS 2 which are required for the use case
- Build your own subsystem on top

→ Reduced effort to validate the custom application by
  - Using certified subsystem
  - Reducing footprint as much as possible
Recap #3

✅ Choose process layout at deploy time
  ⏰ Support from launch to easily configure this

⏳ Event based system providing the infrastructure for fault detection, no tooling yet

✅ Security following the DDS-Security standard
  ⏰ Fine grain configuration

⏳ None of the ROS 2 development is certified
  ✅ But it can interoperate with certified implementations
Questions...

For more information go to: ros2.org