Digital Twin KR3

ROS2 Interfaces

Ahmed Ibrahim Almohamed

Wednesday 11th September, 2024

1 Introduction

The main purpose of this document is to provide a brief overview of the ROS2 Controller layer and how to implement the interfaces of this layer. This is a critical part of this project because it enables us to connect the high level commands from the OPCUA Server with the controller ergo the hardware. The controller layer is responsible for the low level control of the robot. It is the layer that sends the commands to the hardware and receives the feedback from the hardware.

2 Architecture of the Controller Layer

2.1 Overview in the ROS2 Context

As shown in the **Figure 1**, the ROS2 flow starts with a high level node (User or just a normal Node) that sends a high level command to a **3rd party ros2 library (Moveit2)** which is responsible for the motion planning. The 3rd party library sends the planned trajectory the **ROS2-controller** layer which contains a **controller management layer** and a **Resource management layer**. The planned trajectory entries the **controller management layer** which communicates with the **hardware** through the **Resource management layer**. The **Resource management layer** contains user defined interfaces to communicate with the **hardware or the simulation**. [1]

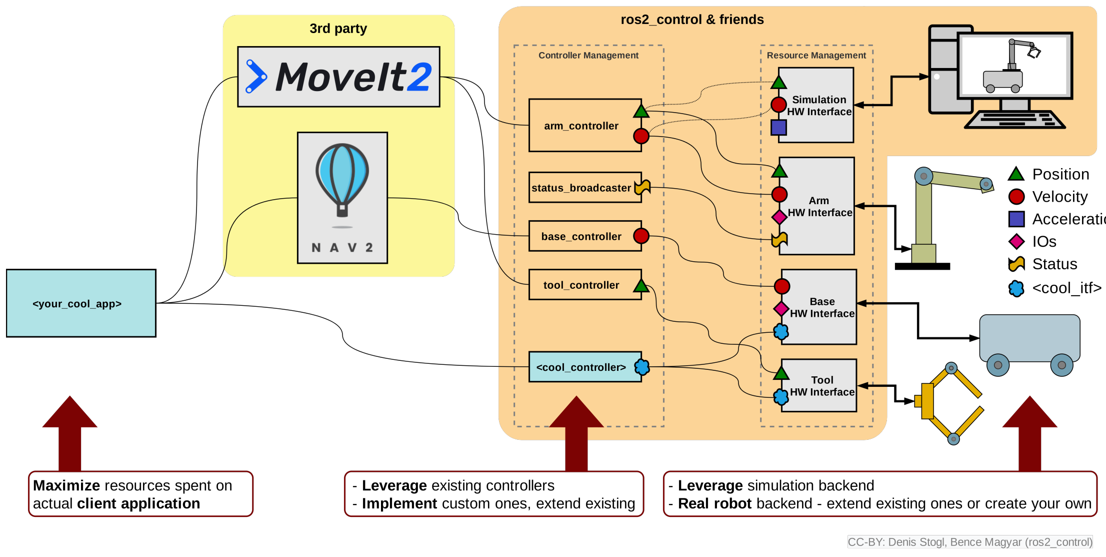

Figure 1: ROS2 control Concept

2.2 ROS2-controller Components

the **Figure 2** is describing the main components of the ROS2 controller , the is a set of controllers "A, B, C", these controllers are mainly a yaml file with parameters that are passed to the controller manager. On the other side the Resource manager collects all the hardware interfaces as plugins libraries and connects them to the controller manager. Also the hardware interfaces are mainly two types , a command interface where the controller manager reads and writes to it and a state interface where the controller manager reads from it. [2]

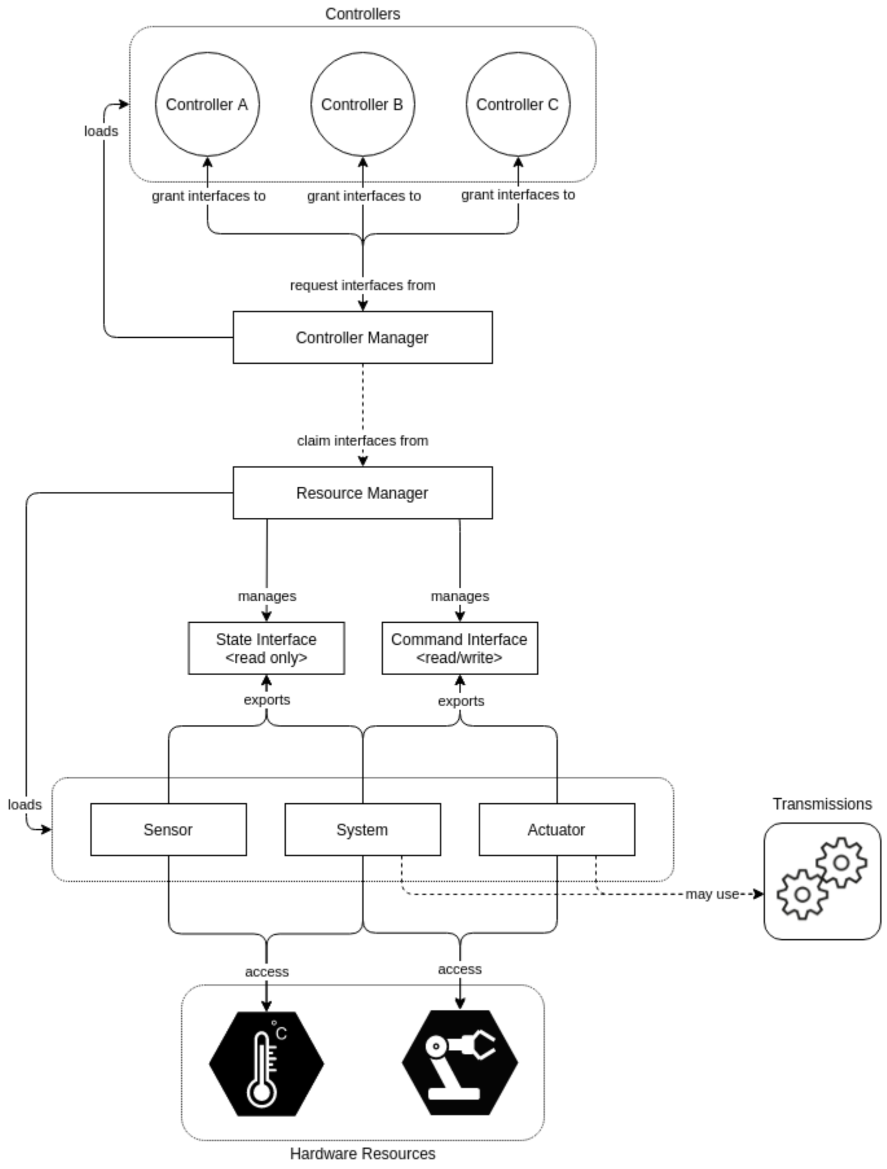

Figure 2: ROS2 control components

3 Type of Controllers for Arm Manipulation

The ROS2 controller package contains a set of controllers that covers the most common robotic applications , such as trajectory tracking, position control, velocity control, and effort control. The **Table 1** [3] contains the controllers used in the ros2 controller package for arm robots.

Controller	**Description**
Admittance Controller	Enables you do zero-force control from a force measured on your TCP. The controller implements ChainedControllerInterface, so it is possible to add another controllers in front of it, e.g., JointTrajectoryController.
Effort Controllers	Collection of controllers that work using the "effort" joint command interface but may accept different joint-level commands at the controller level, e.g. controlling the effort on a certain joint to achieve a set position.
Forward Command Controller	This is a base class implementing a feedforward controller.Specific implementations of this base class can be found in position-controllers ,velocity-controllers ,effort-controllers.
Gripper Controller	Controllers for executing a gripper command action for simple single-dof grippers.
Joint Trajectory Controller	Controller for executing joint-space trajectories on a group of joints. The controller interpolates in time between the points so that their distance can be arbitrary. Even trajectories with only one point are accepted. Trajectories are specified as a set of waypoints to be reached at specific time instants, which the controller attempts to execute as well as the mechanism allows. Waypoints consist of positions, and optionally velocities and accelerations.
PID Controller	The controller can be used directly by sending references through a topic or in a chain having preceding or following controllers. It also enables to use the first derivative of the reference and its feedback to have second-order PID control.
Position Controllers	This is a collection of controllers that work using the "position" joint command interface but may accept different joint-level commands at the controller level.
Velocity Controllers	This is a collection of controllers that work using the "velocity" joint command interface but may accept different joint-level commands at the controller level.

Table 1: ROS 2 Controllers for Robotic Arms and Related Tools

4 Hardware Interfaces

Hardware interfaces are implemented as plugins that follow a lifecycle model similar to ROS2 Lifecycle Nodes. This design allows for better management of hardware resources and more robust error handling.

4.1 Hardware Interface Lifecycle

The lifecycle of a Hardware Interface in ROS2 Control consists of several states:

1. Unconfigured
2. Inactive
3. Active
4. Finalized

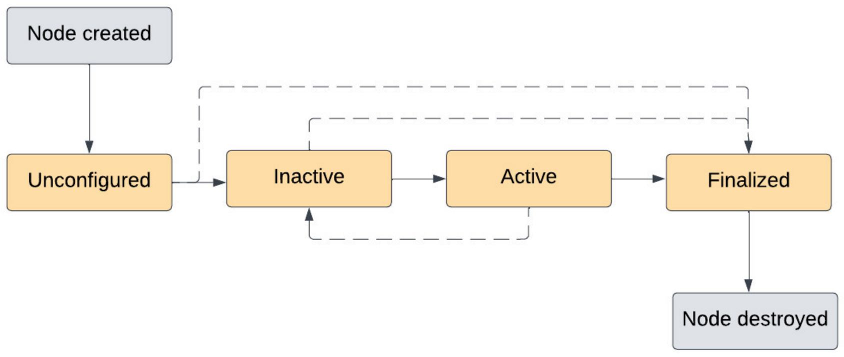

Figure 3: ROS2 Live Node Overview

Each state transition triggers specific callback functions.

- `on_init()`: Called when initializing the hardware interface
- `on_configure()`: Sets up communication with the hardware
- `on_activate()`: Prepares the hardware for operation
- `on_deactivate()`: Stops hardware operation
- `on_cleanup()`: Cleans up resources
- `on_shutdown()`: Prepares for shutdown
- `on_error()`: Handles errors during operation

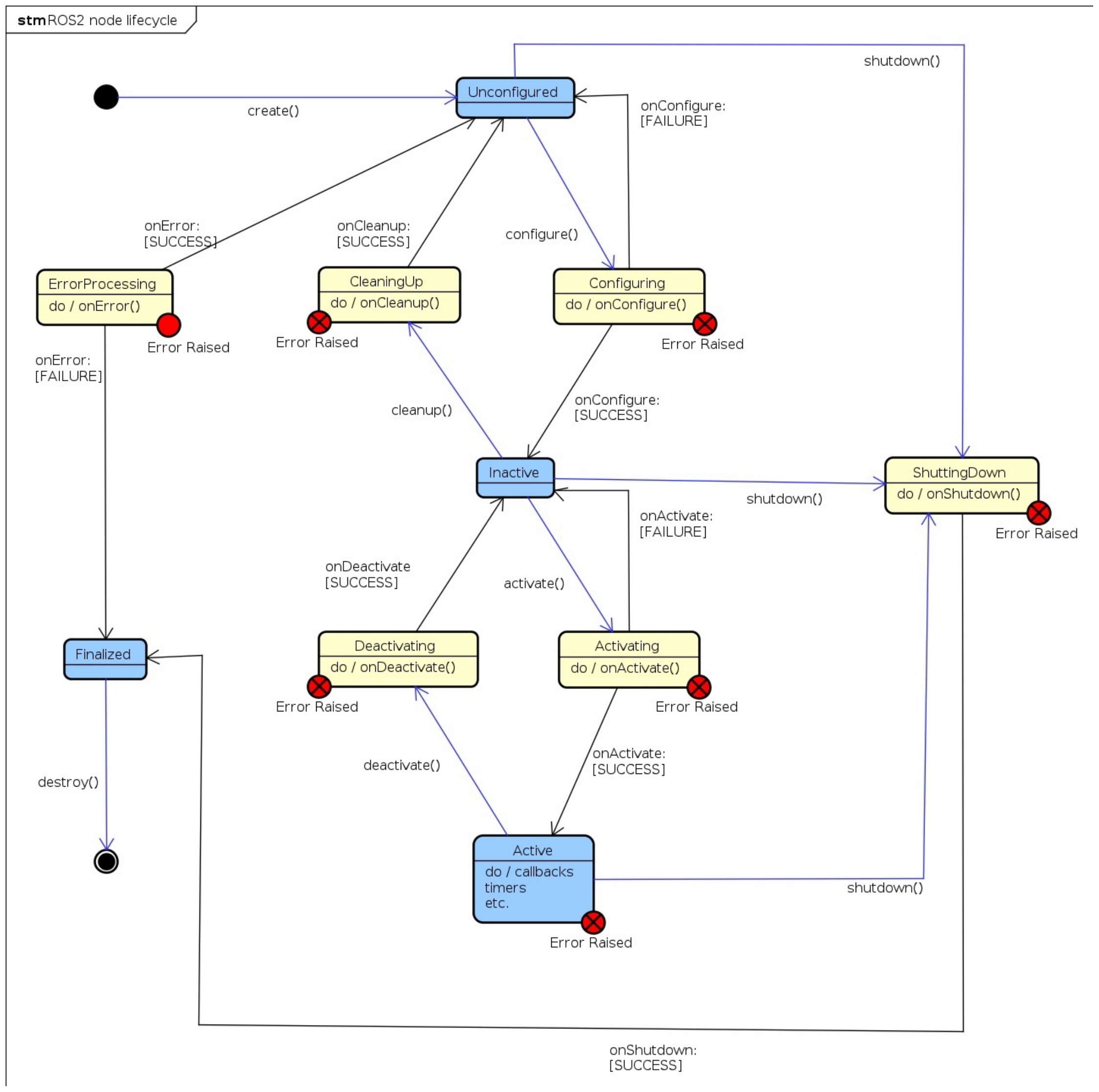

Figure 4: ROS2 Live Node State Machine

4.2 Asynchronous Communication

synchronous communication is crucial for Hardware Interfaces, especially when dealing with real-time control systems or hardware with varying response times. Since KUKAVARPROXY uses TCP, you'll need to implement asynchronous communication to avoid blocking the control loop.

1. **Non-blocking operations**: Asynchronous communication allows the control loop to continue running without waiting for hardware responses, maintaining system responsiveness.

2. **Handling slow hardware**: Some hardware components may have slow update rates or communication delays. Asynchronous operations prevent these delays from affecting the entire control system.

3. **Parallel processing**: Multiple hardware components can be read from or written to simultaneously, improving overall system performance.

4. **Error resilience**: Asynchronous operations can better handle timeouts and communication errors without blocking the entire system.

References

[1] fjp, “Ros 2 controllers overview,” https://fjp.at/posts/ros/ros-control/, 2024, [Online; accessed 13-09-2024].

[2] R. H. et al., “Ros 2 controllers architecture,” https://control.ros.org/humble/doc/getting_started/getting_started.html, 2024, [Online; accessed 13-09-2024].

[3] ——, “Ros 2 controllers documentation,” https://control.ros.org/humble/doc/ros2_controllers/doc/controllers_index.html, 2024, [Online; accessed 13-09-2024].