



## D4.9

# Lifecycle and System Modes Software Release Y2

Grant agreement no.	780785
Project acronym	OFERA (micro-ROS)
Project full title	Open Framework for Embedded Robot Applications
Deliverable number	D4.9
Deliverable name	Lifecycle and System Modes – Software Release Y2
Date	December 2019
Dissemination level	public
Workpackage and task	4.3
Author	Arne Nordmann (Bosch)
Contributors	Ralph Lange (Bosch)
Keywords	micro-ROS, robotics, ROS, microcontrollers, runtime lifecycle, system modes
Abstract	This document provides links to the released software and documentation for deliverable D4.9 <i>Lifecycle and System Modes Software Release Y2</i> of the Task 4.3 <i>Lifecycle and system modes</i> .



# Contents

<b>1</b>	<b>Overview to Results</b>	<b>2</b>
<b>2</b>	<b>Links to Software Repositories</b>	<b>2</b>
<b>3</b>	<b>Annex 1: Webpage on System Modes</b>	<b>3</b>
3.1	Introduction and Goal . . . . .	3
3.2	Requirements . . . . .	4
3.3	Background: ROS 2 Lifecycle . . . . .	4
3.4	Main Features . . . . .	5
3.4.1	Extended Lifecycle . . . . .	5
3.4.2	System Hierarchy and Modes . . . . .	5
3.4.3	Mode Inference . . . . .	5
3.4.4	Mode Manager . . . . .	6
3.5	Roadmap . . . . .	6
3.5.1	2018 . . . . .	6
3.5.2	2019 . . . . .	6
3.5.3	2020 . . . . .	6
3.6	Acknowledgments . . . . .	6

# 1 Overview to Results

This document provides links to the released software and documentation for deliverable D4.9 *Lifecycle and System Modes Software Release Y2* of the Task 4.3 *Lifecycle and system modes*.

**As an entry-point to all software and documentation, we created a dedicated webpage on the micro-ROS website: [https://micro-ros.github.io/system\\_modes/](https://micro-ros.github.io/system_modes/)**

The annex includes a copy of this webpage and copies of the major documentation files of this software release.

## 2 Links to Software Repositories

The parser for the system modes model and the mode inference mechanisms (including the extended lifecycle) are provided in a repository named `system_modes`, together with two ROS nodes for mode management and mode monitoring.

- Git repository: [https://github.com/micro-ROS/system\\_modes](https://github.com/micro-ROS/system_modes)  
Package name: `system_modes`  
Package path: `./system_modes`  
Fixed bugs: [#2](#), [#3](#), [#7](#)

Demo of system modes concept and implementation in same repository:

- Git repository: [https://github.com/micro-ROS/system\\_modes](https://github.com/micro-ROS/system_modes)  
Package name: `system_modes_examples`  
Package path: `./system_modes_examples`

The system modes package and the system modes example package were bloomed and released for the ROS 2 distributions Crystal, Dashing, and Eloquent:

- Git repository: [https://github.com/micro-ROS/system\\_modes-release](https://github.com/micro-ROS/system_modes-release)  
Releases: [ROS 2 Crystal \(0.1.3-1\)](#), [ROS 2 Dashing \(0.1.4-1\)](#), [ROS 2 Eloquent \(0.1.5-1\)](#)

The ROS diagnostics aggregator package was ported to ROS 2 to enable integration with system modes in ROS 2 systems:

- Git repository: <https://github.com/ros/diagnostics>  
Package name: `diagnostic_aggregator`  
Package path: `./diagnostic_aggregator`  
Pull request: [ros2\\_diagnostics/pull/2](#)

Documentation of the system modes package on the micro-ROS website was updated to cover the latest developments:

- Git repository: <https://github.com/micro-ROS/micro-ROS.github.io>  
Commits: [#c5504fa](#), [#c07b172](#)

### 3 Annex 1: Webpage on System Modes

Content of [https://micro-ros.github.io/system\\_modes/](https://micro-ros.github.io/system_modes/) from 5th December 2019.

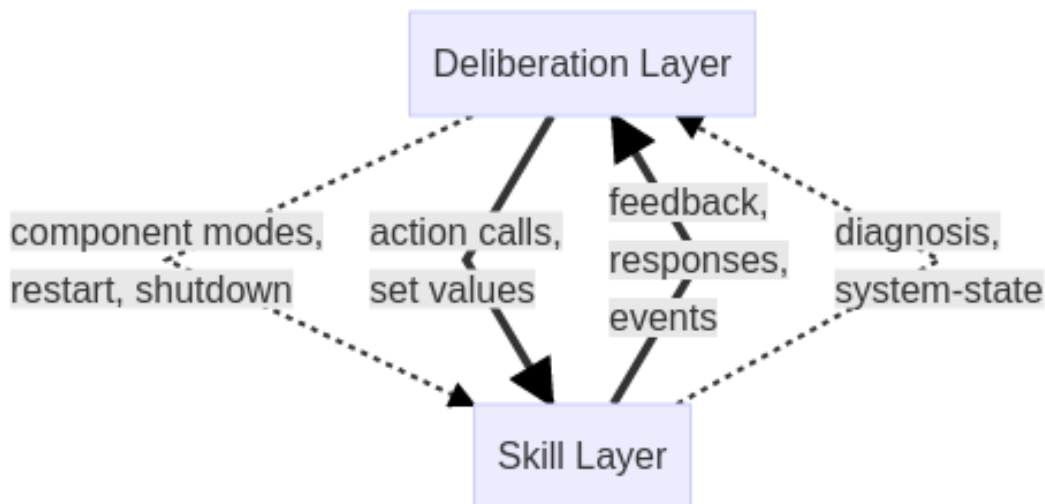
#### 3.1 Introduction and Goal

Modern robotic software architectures often follow a layered approach. The layer with the core algorithms for SLAM, vision-based object recognition, motion planning, etc. is often referred to as *skill layer* or *functional layer*. To perform a complex task, these skills are orchestrated by one or more upper layers named *executive layer* and *planning layer*. Other common names are *task and mission layer* or *deliberation layer(s)*. In the following, we used the latter term.

We observed three different but closely interwoven aspects to be handled on the deliberation layer:

1. **Task Handling:** Orchestration of the actual task, the *straight-forward, error-free* flow.
2. **Contingency Handling:** Handling of task-specific contingencies, e.g., expectable retries and failure attempts, obstacles, low battery.
3. **System Error Handling:** Handling of exceptions, e.g., sensor/actuator failures.

The mechanisms being used to orchestrate the skills are service and action calls, re-parameterizations, set values, activating/deactivating of components, etc. We distinguish between *function-oriented calls* to a running skill component (set values, action queries, etc.) and *system-oriented calls* to individual or multiple components (switching between component modes, restart, shutdown, etc.).

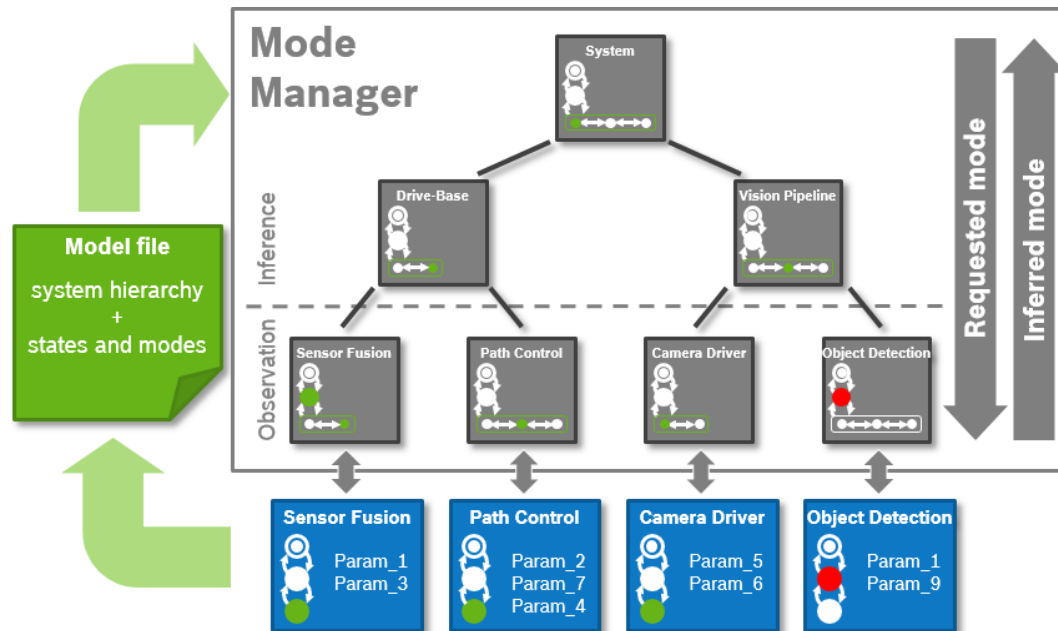


Analogously, we distinguish between *function-oriented notifications* from the skill layer in form a feedback on long-running service calls, messages on relevant events in the environment, etc. and *system-oriented notifications* about component failures, hardware errors, etc.

Our observation is that interweaving of task handling, contingency handling, and system error handling generally leads to a high complexity of the control flow on the deliberation layer. Yet, we hypothesize that this complexity can be reduced by introducing appropriate abstractions for system-oriented calls and notifications.

Therefore, our **goal** within this work is to provide suitable abstractions and framework functions for (1.) system runtime configuration and (2.) system error and contingency diagnosis, to reduce the effort for the application developer of designing and implementing the task, contingency and error handling.

This goal is illustrated in the following example architecture, which is described and managed based on a model file:



The main features of the approach are (detailed in the remainder):

1. *Extended Lifecycle*: Extensible concept to specify the runtime states of components, i.e ROS 2 lifecycle nodes.
2. *System Hierarchy and Modes*: Modeling approach for specifying a ROS system in terms of its system hierarchy and *system modes*, i.e. different (sub-)system configurations.
3. *Mode Manager*: A module to manage and change the system runtime configuration.
4. *Mode Inference*: A module for deriving the entire system state and mode from observable system information, i.e. states, modes, and parameters of its components.
5. *Diagnostics*: Diagnosis module for deriving relevant information from the operating systems, the hardware and the functional components.

## 3.2 Requirements

The list of requirements is maintained in the doc folder of the micro-ROS system modes repository, at: [https://github.com/micro-ROS/system\\_modes/blob/master/system\\_modes/doc/requirements.md](https://github.com/micro-ROS/system_modes/blob/master/system_modes/doc/requirements.md)

## 3.3 Background: ROS 2 Lifecycle

Our approach is based on the ROS 2 Lifecycle. The primary goal of the ROS 2 lifecycle is to allow greater control over the state of a ROS system. It allows consistent initialization, restart and/or

replacing of system parts during runtime. It provides a default lifecycle for managed ROS 2 nodes and a matching set of tools for managing lifecycle nodes.

The description of the concept can be found at: [http://design.ros2.org/articles/node\\_lifecycle.html](http://design.ros2.org/articles/node_lifecycle.html)  
The implementation of the Lifecycle Node is described at:  
<https://index.ros.org/doc/ros2/Managed-Nodes/>.

## 3.4 Main Features

### 3.4.1 Extended Lifecycle

In micro-ROS, we extend the ROS 2 lifecycle by allowing to specify modes, i.e. substates, specializing the *active* state based on the standard ROS 2 parameters mechanism. We implemented this concept based on rcl and rclcpp for ROS 2 and micro-ROS.

Documentation and code can be found at:  
[github.com:system\\_modes/README.md#lifecycle](https://github.com/system_modes/README.md#lifecycle)

### 3.4.2 System Hierarchy and Modes

We provide a modeling concept for specifying the hierarchical composition of systems recursively from nodes and for specifying the states and modes of systems and (sub-)systems with the extended lifecycle, analogously to nodes. This system modes and hierarchy (SMH) model also includes an application-specific the mapping of the states and modes along the system hierarchy down to nodes.

The description of this model can be found at:  
[github.com:system\\_modes/README.md#system-modes](https://github.com/system_modes/README.md#system-modes)

A simple example is provided at:  
[github.com:system\\_modes\\_examples/README.md#example-mode-file](https://github.com/system_modes_examples/README.md#example-mode-file)

### 3.4.3 Mode Inference

The mode inference infers the entire system states (and modes) based on the lifecycle states, modes, and parameter configuration of its components, i.e. the ROS 2 lifecycle nodes. It parses the SMH model and subscribes to lifecycle/mode change requests, lifecycle/mode changes, and parameter events.

Based on the lifecycle change events it knows the *actual* lifecycle state of all nodes. Based on parameter change events it knows the *actual* parameter values of all nodes, which allows inference of the *modes* of all nodes based on the SMH model. Based on the SMH model and the inferred states and modes of all nodes, states and modes of all (sub-)systems can be *inferred* bottom-up along the system hierarchy. This can be compared to the latest *requested* states and modes to detect a deviation.

The documentation and code can be found at:  
[github.com:system\\_modes/README.md#mode-inference](https://github.com/system_modes/README.md#mode-inference)

The mode inference can be best observed in the mode monitor, a console-based debugging tool, see:  
[github.com:system\\_modes/README.md#mode-monitor](https://github.com/system_modes/README.md#mode-monitor)

### 3.4.4 Mode Manager

Building upon the *Mode Inference* mechanism, the mode manager provides additional services and topics to *manage and adapt* system states and modes according to the specification in the SMH model.

The documentation and code can be found at:

[github.com:system\\_modes/README.md#mode-manager](https://github.com/system_modes/README.md#mode-manager)

A simple example is provided at:

[github.com:system\\_modes\\_examples/README.md#setup](https://github.com/system_modes_examples/README.md#setup)

## 3.5 Roadmap

### 3.5.1 2018

- Requirements for behavior system composition in robotics systems
- Modeling concept to specify system hierarchy as well as system modes of systems, subsystems, and their mapping along the system hierarchy down to nodes
- Python prototype of system modes concept (ROS-independent)

### 3.5.2 2019

- Extended lifecycle concept and implementation for ROS 2 C++
- Mode inference and mode manager for ROS 2 C++
- ROS diagnostics port from ROS 1 to ROS 2
- Concept for bottom-up system modes rules

### 3.5.3 2020

- Diagnostics framework for micro-ROS, interoperating with ROS 2 diagnostics
- MCU-specific diagnostics functions for resource usage on RTOS layer, latencies, statistics from middleware, etc.
- Integration of mode manager with real-time executor and/or roslaunch
- Lightweight concept for specifying error propagation and recovery mechanisms

*Note: The extension of the ACTIVE state by modes (substates) was originally planned for 2020 but brought forward in 2018.*

## 3.6 Acknowledgments

This activity has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement n° 780785).