Linux-based computer

- 300 MHz Clock Frequency
- 1 MB SRAM
- AUTOSAR Classic

Sensors, Actuators
Micro-ROS Architecture

C API

rcl + rclc: Convenience functions, executor, node graph, …

rmw – middleware interface

Adapter

Micro XRCE-DDS Client

POSIX interface

Additional drivers

FreeRTOS, Zephyr, NuttX, …

microcontroller

UART, IP, Bluetooth, …

ROS 2 stack

micro-ROS Agent

DDS

Linux

microprocessor
Modified Architecture

- C API
- `rmw` – middleware interface
- Adapter
- Micro XRCE-DDS Client
  - CLib + POSIX compatibility layer
  - AUTOSAR Classic (here with Rexroth’s BODAS API)
- microcontroller

- Application component
- ... Application component

- New Executor

- `rcl` + `rclc`

- CAN as custom transport in Micro XRCE-DDS

- ROS 2 stack
  - micro-Ros Agent
  - DDS
  - Linux

- microprocessor
Challenge 1: Mapping ROS Execution to AUTOSAR Tasks

ROS 2
POSIX threads

AUTOSAR Classic
Fixed Periodic Task Scheduling

Node 1 Executor
cb₁

Node 2 Executor
cb₂

cb₃
Solution: Dispatching Executor

ROSClassic

Available in github.com/ros2/rclc (for Rolling)
Challenge 2: Handling blocking initializations in rcl and rmw

On POSIX

Client thread \textit{waits} for Agent’s reply

On AUTOSAR Classic

Periodic tasks \textit{cannot wait} in stack
Solution: Refactoring into state machine

State machine

- Create entity
- Sent request
- Received response

Client

Agent

time
Solution: Refactoring into state machine

State machine

Create entity

Sent request

Received response

Client

Agent

time
Solution: Refactoring into state machine

State machine

Create entity

Sent request

Received response

Client

Agent

time
Solution: Refactoring into state machine

State machine

Create entity

Sent request

Received response

Client

Agent

time
Solution: Refactoring into state machine

State machine

Create entity → Sent request → Received response

Client → Agent

Client

Client

time
How to model the performance of a micro-ROS application?

Application Model
- #subscriptions
- message size
- data rate

Micro-ROS Model

Hardware Model
- Execution time
Demonstrator with BODAS Controller by Bosch Rexroth
## Results

- Runtime overhead by micro-ROS stack is < 3%
  - ... despite quick user code callbacks

- Model estimates response times with error of ≈6%

<table>
<thead>
<tr>
<th></th>
<th>Real HW</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>min latency [us]</td>
<td>970</td>
<td>1029</td>
</tr>
<tr>
<td>avg latency [us]</td>
<td>10320</td>
<td>n/a</td>
</tr>
<tr>
<td>max latency [us]</td>
<td>19978</td>
<td>20070</td>
</tr>
</tbody>
</table>
Ported micro-ROS stack to AUTOSAR-based platform

- rclc Dispatching Executor to map execution model
- Refactored blocking initializations into state machines
- Support for classical CAN as custom transport
- micro-ROS performance model in Amalthea

Many thanks to Kaiwalya Kalyan Belsare and Suraj Rao Bappanadu for the contributions in their Master’s theses!

Learn more?

- Jan.Staschulat@de.bosch.com | github.com/JanStaschulat
- Ralph.Lange@de.bosch.com | github.com/ralph-lange

... also see upcoming chapter “micro-ROS” in Springer’s ROS Book Vol. 7