A Model-Based Methodology for Automated Verification of ROS 2 Systems

Lukas Dust
lukas.dust@mdu.se
Mälardalen University
Västerås, Sweden

Mikael Ekström
mikael.ekstrom@mdu.se
Mälardalen University
Västerås, Sweden

Rong Gu
rong.gu@mdu.se
Mälardalen University
Västerås, Sweden

Saad Mubeen
saad.mubeen@mdu.se
Mälardalen University
Västerås, Sweden

Cristina Seceleanu
cristina.seceleanu@mdu.se
Mälardalen University
Västerås, Sweden
Introduction

Robot Operating System (ROS)

- Open Source Middleware for fast prototyping and development of distributed (robotic) systems
- Often strict timing requirements
- Added Real-Time capabilities by adding Data Distribution System (DDS) to ROS 2
- ROS 2 is used in academia and industry and finds application in real world systems like warehouse logistics
Especially in distributed systems, real-time capabilities are influenced by multiple system components, including communication, task scheduling and execution.
Verification in various forms is needed
Formal Methods provide mathematical and rigorous analysis during system design time

Model-Based Verification:
Exhaustive checking of the whole state space based on formal models

UPPAAL
- State of the art tool supporting modeling, simulation and model checking for an extension of TA (UTA)
- UTA are modeled as templates that can be composed to a system

Drawback of Formal Modeling:
- complicated modeling process
- steep learning curve for syntax, semantics and the formal modeling language

→ Trial and Error Approaches are predominant in cyberphysical systems
Pattern-Based Verification of ROS 2

Pattern-Based Verification of Buffer Sizes and Callback Latency


Pattern-Based Verification of ROS 2

```c
// Simulation Time
const int StopTime = 100;

// Release Times for WallTimeCallbacks
const int releasesH[MAXX]={0,32,63,0,0,0,0,0,0,0};
const int releasesM[MAXX]={0,32,0,0,0,0,0,0,0,0};
const int releasesL[MAXX]={0,32,0,0,0,0,0,0,0,0};
const int releasesSH[MAXX]={0,45,0,0,0,0,0,0,0,0};
const int releasesSM[MAXX]={0,45,0,0,0,0,0,0,0,0};
const int releasesSL[MAXX]={0,0,0,0,0,0,0,0,0,0};

// Executors
ExecutorV1 = ExecutorV1(StopTime);
ExecutorV2 = ExecutorV2(StopTime);

// Callbacks
T0 = PeriodicCallback(0, 5, 13, TIMER, 0, 1);
H = WallTimeCallback(0, 5, 3, releasesH, SUBSCRIBER,10);
M = WallTimeCallback(1, 5, 2, releasesL, SUBSCRIBER,10);
L = WallTimeCallback(2, 5, 2, releasesM, SUBSCRIBER,10);
SH = WallTimeCallback(0, 5, 2, releasesSH, SERVICE,10);
SM = WallTimeCallback(1, 5, 2, releasesSM, SERVICE,10);
SL = WallTimeCallback(2, 5, 1, releasesSL, SERVICE,10);

// System Definition
system ExecV1 < H, M, L, SH, SM, SL, T0;
//system ExecV2 < H, M, L, SH, SM, SL, T0;
```

Node Executor

- **T0**
- **H**
- **M**
- **L**
- **SH**
- **SM**
- **SL**

Needed Parameters: Callback Types, Execution Times, Release Times, Buffer Sizes, Priority

Problem Formulation

Goal: Application of Software Engineering Methods and Tools to achieve automation and simplification
Architectural Overview

Currently:
Manual Extraction
Architectural Overview – Parameter extraction

ROS 2 Tracing: Low-Overhead Framework for Real-Time Tracing of ROS 2

- Natively in ROS 2 distributions
- Based on LTTNG tracing with tracepoints in the ROS 2 source code
- Contains a ROS 2 Data Model in form of a python pandas table

Architectural Overview

Parsing is needed
Architectural Overview – Modeling Layer

- Combination of Metamodels and Model Transformations in a modular approach

- Adaptability and traceability, each metamodel focuses on specific parameters and visualization to improve understandability

- Automated Transformations between the models

→ Used Tools: Eclipse, EMF, QVT-O and Acceleo


Architectural Overview – Modeling Layer

- **MM1**: Representation of the trace data for early analysis and simple parsing
- **MM2**: Verification Metamodel, abstraction of the needed parameters for verification
- **MM3**: System Metamodel, systems description and abstraction, hierarchical model to analyze and understand system design
  - An understandable system model for ROS 2 developers
Architectural Overview
Workflow

Start A → System Implementation → Execution and Tracing → ROS 2 Data Model

Generation / Manual Parameter Adaption

Start B → EMF System Model → EMF Data Model

Feedback

Generation

EMF Verification Model

Generation

Verification Trace / Query Result

Execution

UPPAAL Verification Code

ROS 2

ROS2_Tracing Trace_Analysis

Model Editor ECLIPSE EMF, QVT-O, Acceleo

Verification Tool (UPPAAL)
Proof of Concept

```python
# Process
events = load_file(path)
handler = Ros2Handler.process(events)
#handler.data.print_data()
data_util = Ros2DataModelUtil(handler.data)
handler.data.print_xml('trace_generated.ros2datamodel.mm')
```
Proof of Concept
Proof of Concept

```c
// Simulation Time
const int StopTime = 100;

// Take away comment to start verification of executor
*/

// Executor minimal_subscriber

// Release Times for WallTimeCallbacks
const int releasesSUBSCRIBER0[MAXX]={58,78,0,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER1[MAXX]={53,73,0,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER2[MAXX]={48,68,0,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER3[MAXX]={43,0,0,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER4[MAXX]={0,0,0,0,43,43,43,43,43,43,43};

// Executor
ExV1 = ExecutorExV1(StopTime);

// Callbacks
TIMER0 = PeriodicCallback(0,5,20,TIMER,0,1);
TIMER1 = PeriodicCallback(1,5,230,TIMER,0,1);
SUBSCRIBER0 = WallTimeCallback(0,5,2,releasesSUBSCRIBER0,SUBSCRIBER1,10);
SUBSCRIBER1 = WallTimeCallback(1,5,2,releasesSUBSCRIBER1,SUBSCRIBER1,10);
SUBSCRIBER2 = WallTimeCallback(2,5,2,releasesSUBSCRIBER2,SUBSCRIBER2,10);
SUBSCRIBER3 = WallTimeCallback(3,5,1,releasesSUBSCRIBER3,SUBSCRIBER1,10);
SUBSCRIBER4 = WallTimeCallback(4,5,10,releasesSUBSCRIBER4,SUBSCRIBER1,10);

// System Definition
system ExV1 < TIMER0,TIMER1,SUBSCRIBER0,SUBSCRIBER1,SUBSCRIBER2,SUBSCRIBER3,SUBSCRIBER4;
```
Contributions

1. A novel methodology for model-based verification of ROS 2 systems proposing a potential toolchain including UPPAAL, Eclipse and ROS 2 tracing

2. Inclusion of metamodels to capture system structure and support verification activities

3. Automated model-to-model transformations from generated ROS 2 execution traces to Ecore models and automated transformations of Ecore models to UPPAAL models

4. Demonstration of the main workflow in form of a proof of concept implementation covering partial aspects of the toolchain
Future Work

- Some more beneficial trace points can be added to ROS 2 tracing

- Implementation of the missing transformations from the prototype
  - Implementation of requirements management
  - ROS 2 code generation

- Formal proof of the models, further improvements to the verification

- Inclusion of additional verification properties
Questions

Contact:
Email: lukas.dust@mdu.se
Website: www.es.mdu.se/staff/4857-Lukas_Dust
Phone: +460736620999