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

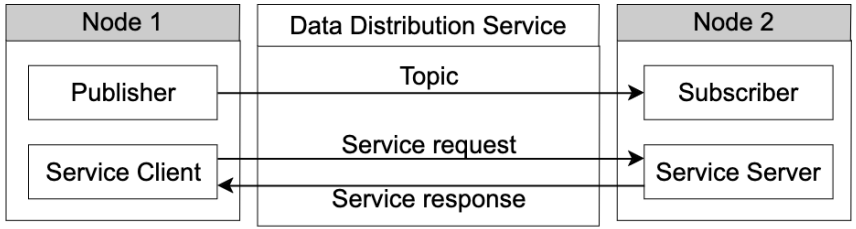
Introduction

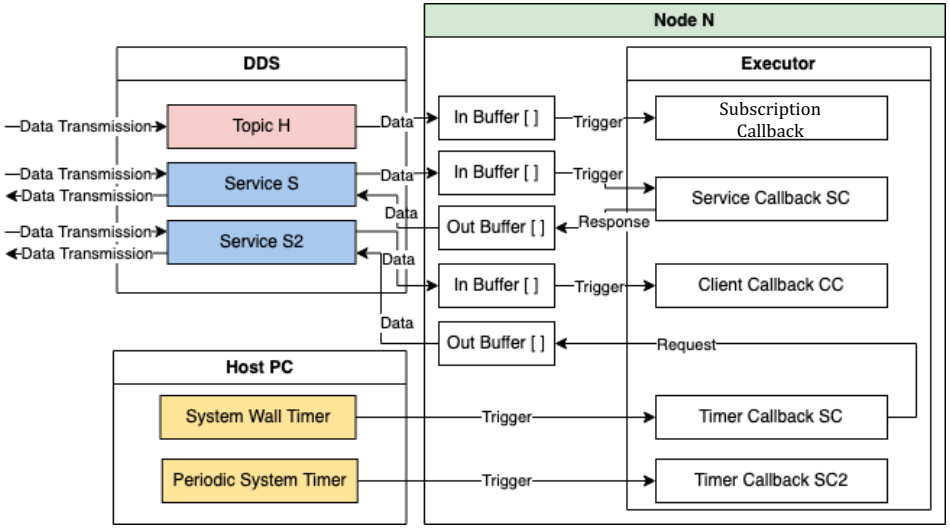
ROS 2 open robotics

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

ROS 2 Concept





- *Especially in distributed systems, real-time capabilities are influenced by multiple system components, including communication, task scheduling and execution.*

Formal Verification and UPPAAL

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

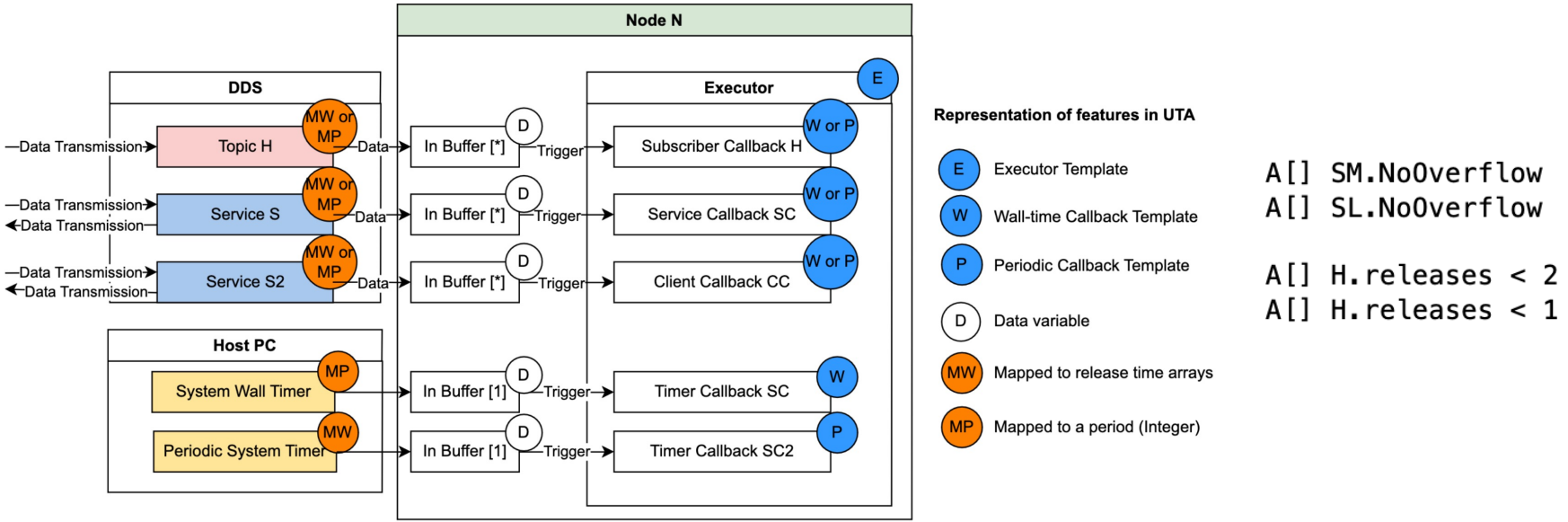
UPPAAL
- State of the art tool supporting modeling, simulation and model checking for an extension of TA (UTA)
- UTA are modeled as templates that can be composed to a system

Drawback of Formal Modeling:
- complicated modeling process
- steep learning corve 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

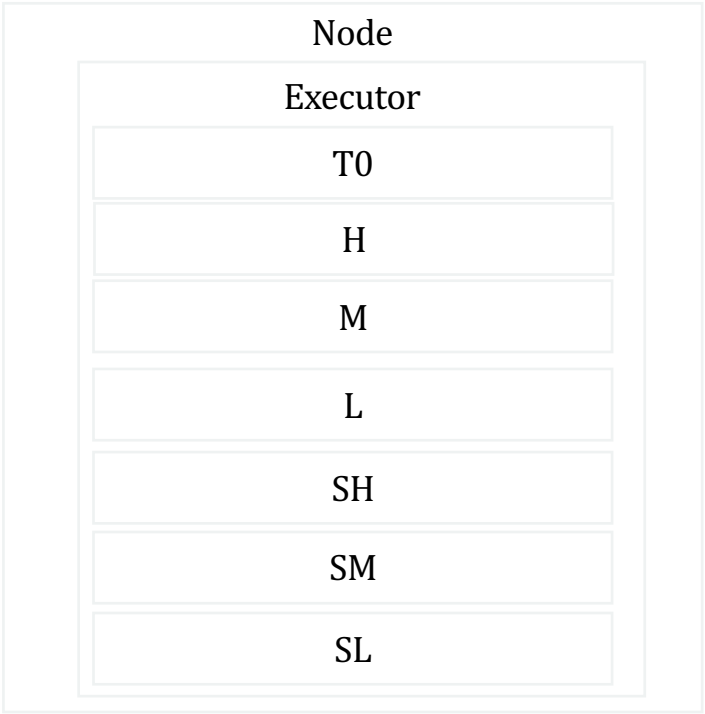


Needed Parameters: System Design, Execution Times, Release Times, Buffer Sizes, Priority

[13] Lukas Dust, Rong Gu, Cristina Seceleanu, Mikael Ekström, and Saad Mubeen. 2023. Pattern-Based Verification of ROS 2 Nodes Using UPPAAL. In *International Conference on Formal Methods for Industrial Critical Systems*. Springer, 57–75.

Pattern-Based Verification of ROS 2

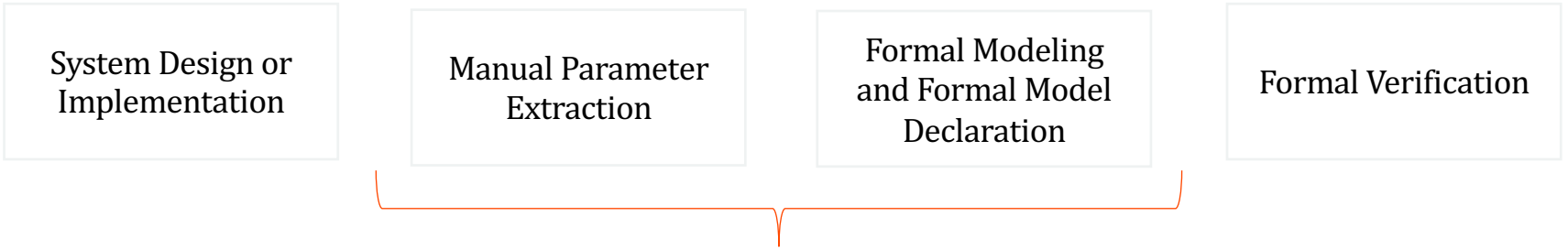
```
// Simulation Time
const int StopTime = 100;

// 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
ExecV1 = ExecutorV1(StopTime);
ExecV2 = 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;
```



[13] Lukas Dust, Rong Gu, Cristina Seceleanu, Mikael Ekström, and Saad Mubeen. 2023. Pattern-Based Verification of ROS 2 Nodes Using UPPAAL. In *International Conference on Formal Methods for Industrial Critical Systems*. Springer, 57–75.

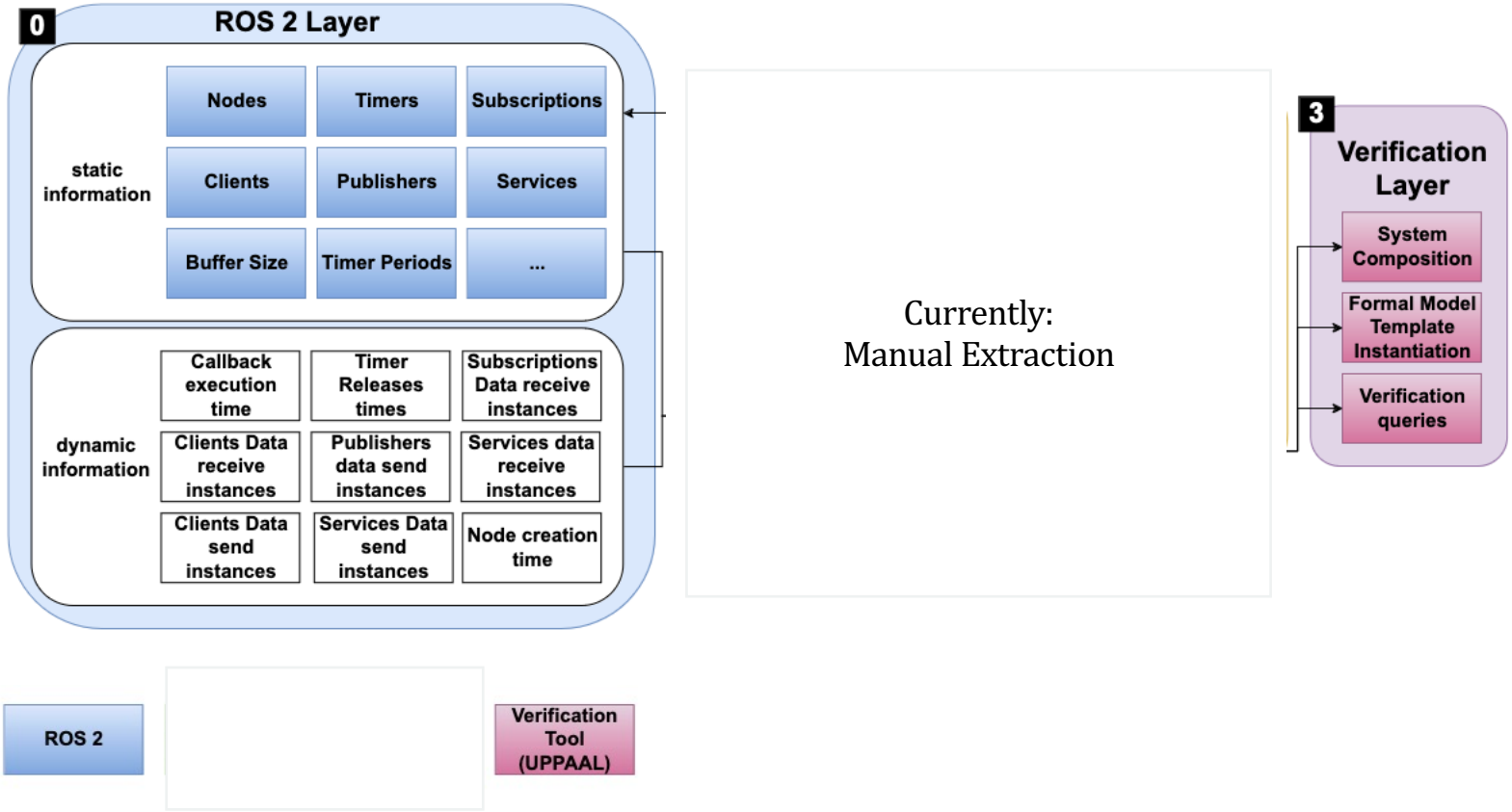
Needed Parameters: Callback Types, Execution Times, Release Times, Buffer Sizes, Priority

Problem Formulation

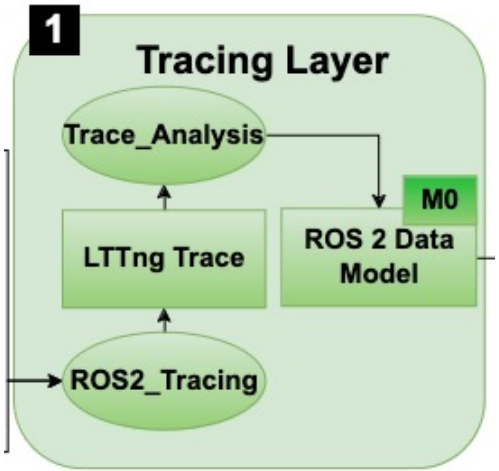


Goal: Application of Software Engineering Methods and Tools to achieve **automation and simplification**

Architectural Overview



Architectural Overview – Parameter extraction

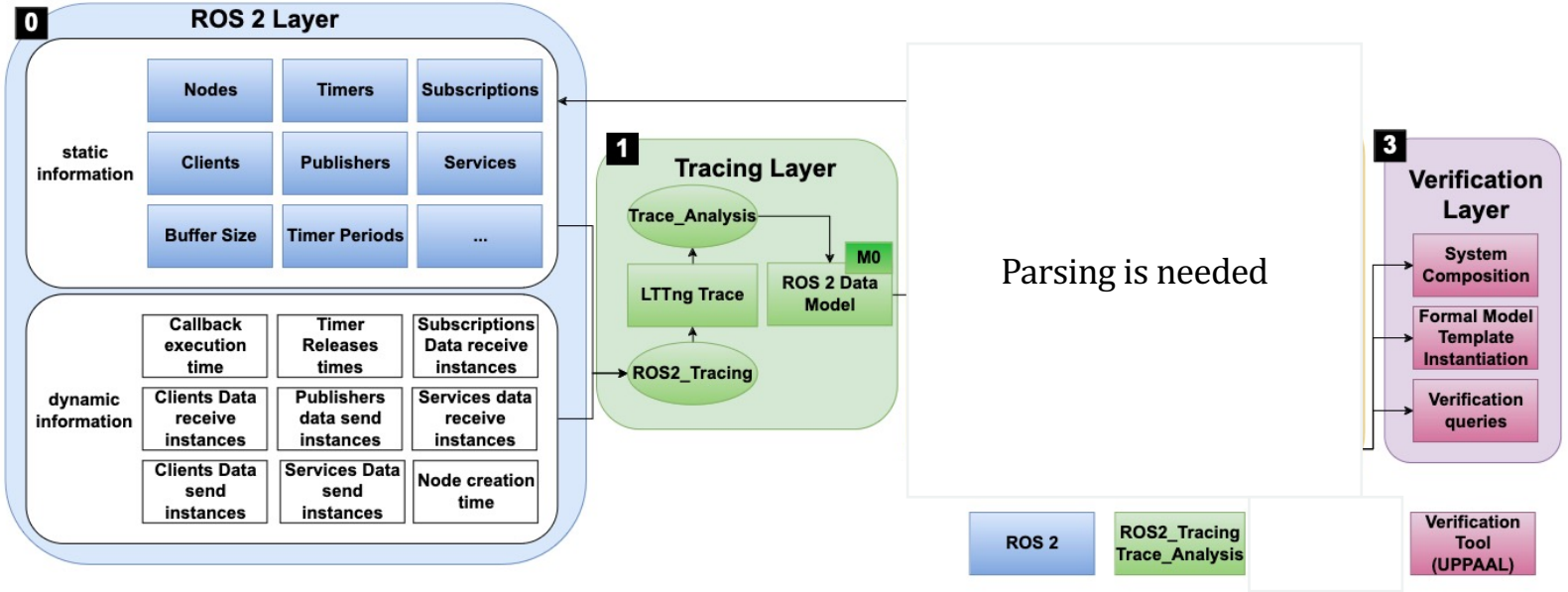


ROS 2 Tracing: Low-Overhead Framework for Real-Time Tracing of ROS 2

- 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

[6] Christophe Bédard, Ingo Lütkebohle, and Michel Dagenais. 2022. ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2. *IEEE Robotics and Automation Letters* 7, 3 (2022), 6511–6518.

Architectural Overview



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

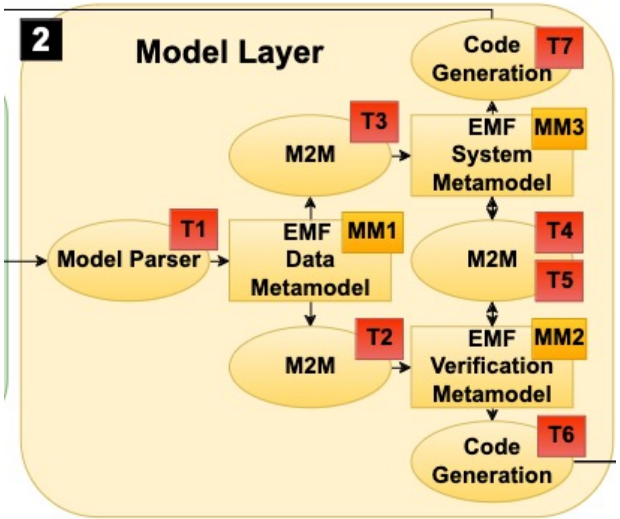
→ Used Tools: Eclipse, EMF, QVT-O and Acceleo

[1] 2011. Acceleo - Transforming Models into Code.

[21] OMG. 2011. MOF 2.0 Query/View/Transformation Spec. V1.1.

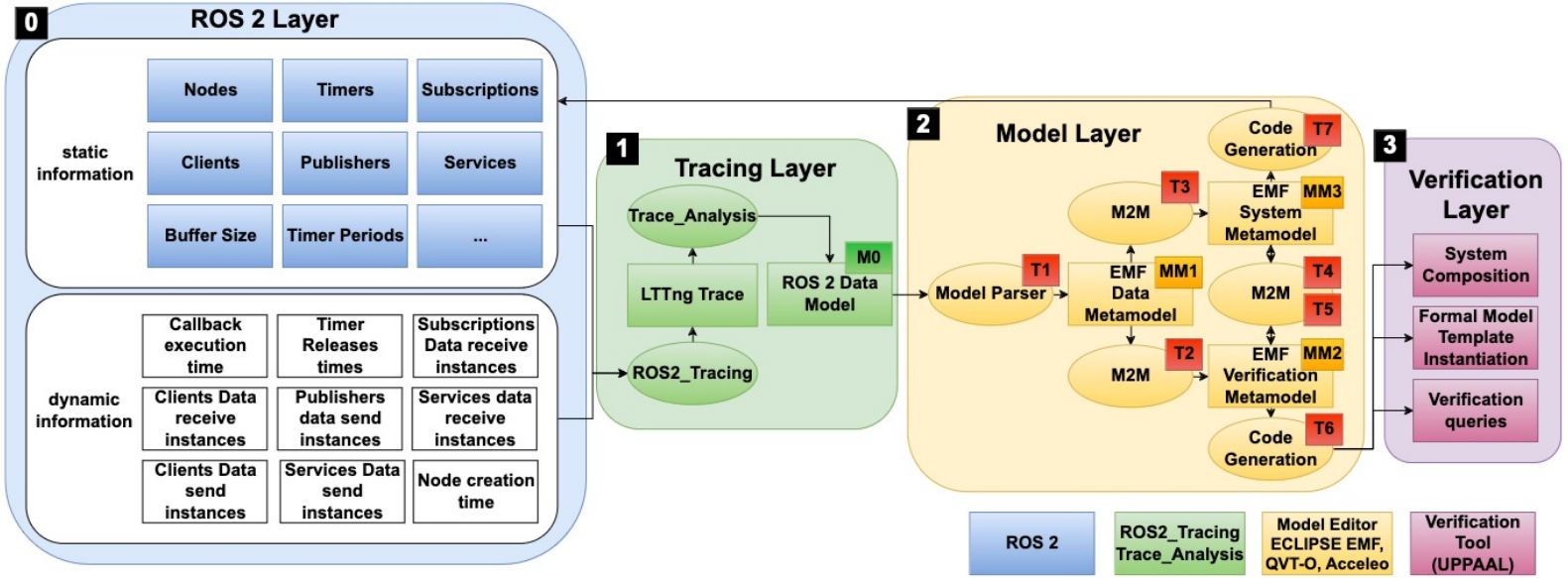
[27] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework.* Pearson Education.

Architectural Overview – Modeling Layer

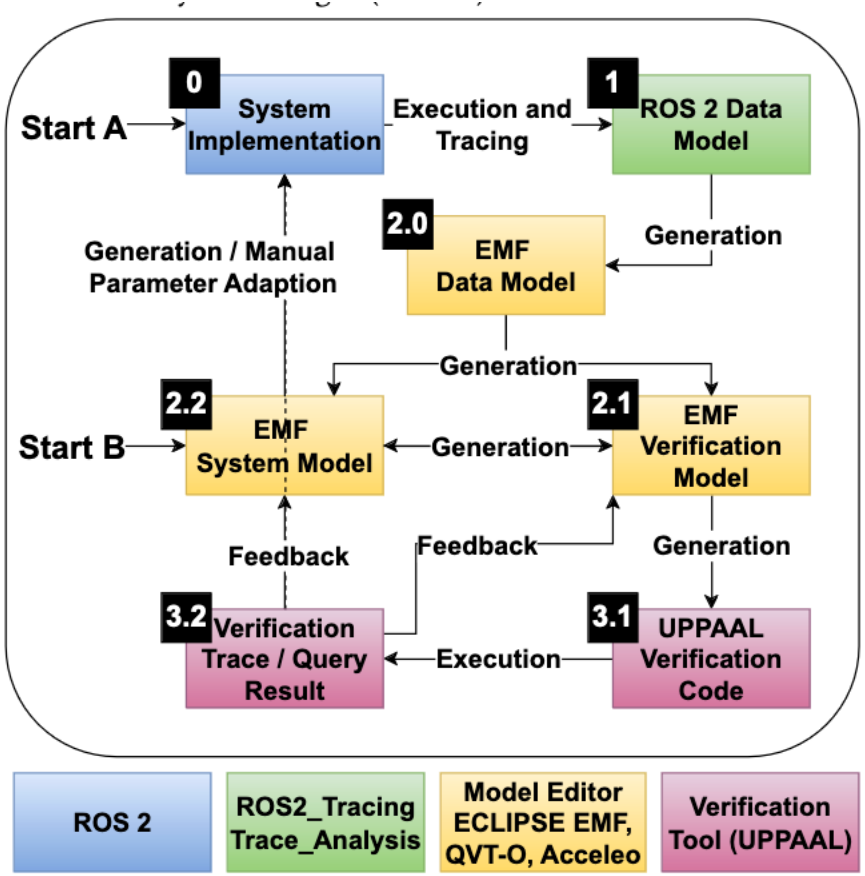


- MM1: Representation of the trace data for early analysis and simple parsing
- MM2: Verification Metamodel, abstraction of the needed parameters for vericication
- MM3: System Metamodel, systems description and abstraction, hierarchical model to analyze and understand system design
  → An understandeable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

Architectural Overview

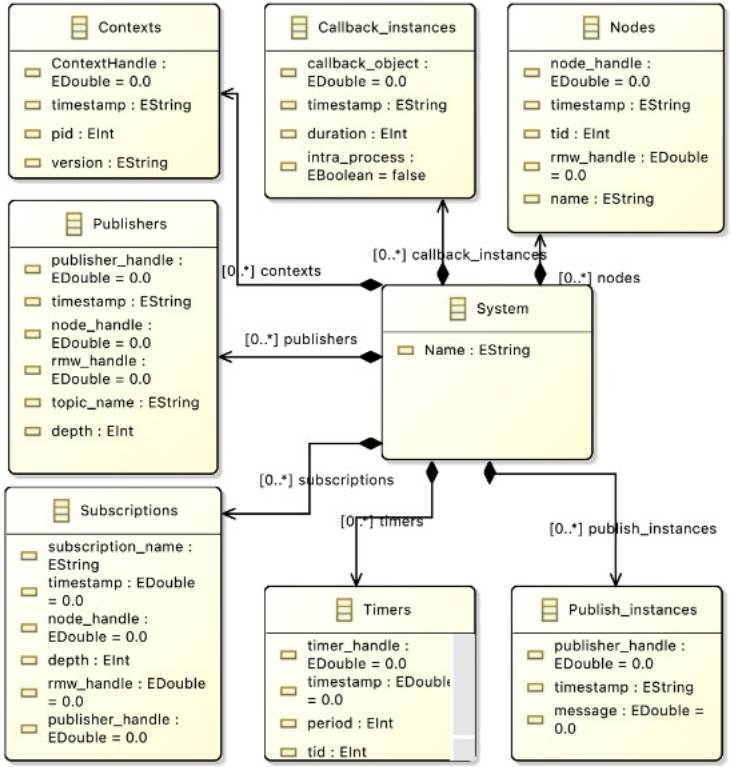


Workflow

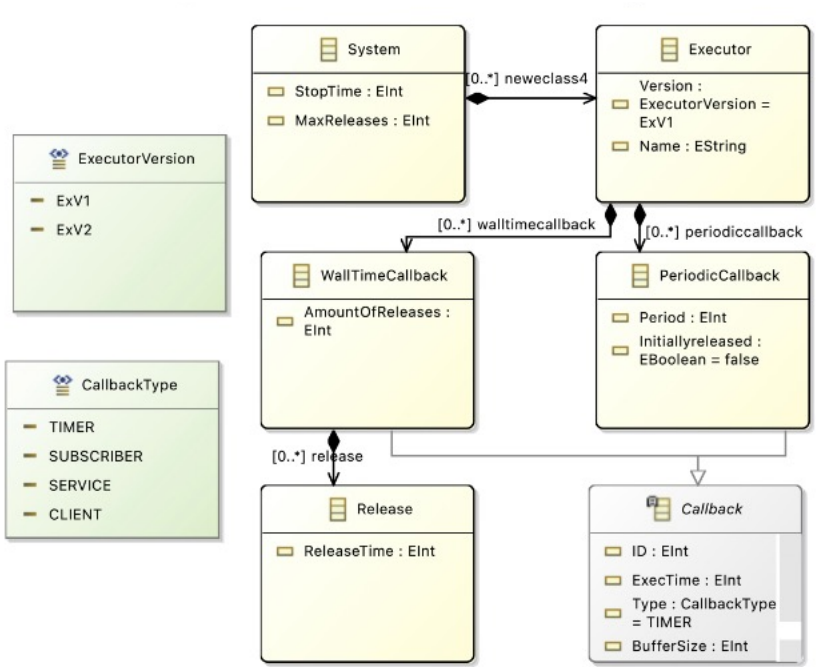

0
System Implementation
Start A
Execution and Tracing
1
ROS 2 Data Model
Generation
2.0
EMF Data Model
Generation / Manual Parameter Adaption
Generation
2.2
EMF System Model
Start B
Generation
2.1
EMF Verification Model
Feedback
Feedback
Generation
3.2
Verification Trace / Query Result
Execution
3.1
UPPAAL Verification Code
ROS 2
ROS2_Tracing Trace_Analysis
Model Editor ECLIPSE EMF, QVT-O, Acceleo
Verification Tool (UPPAAL)

Proof of Concept

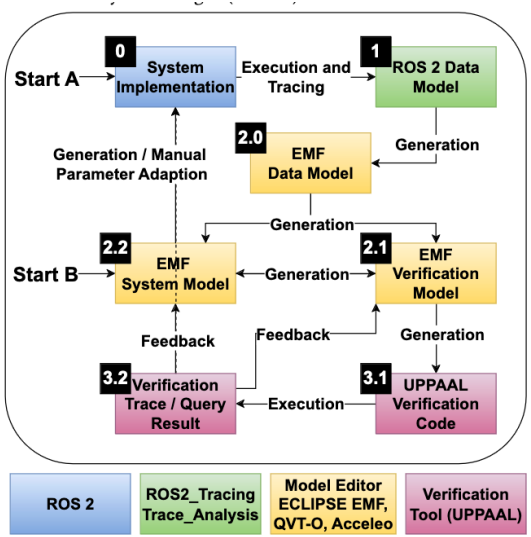
Data Metamodel



Verification Metamodel

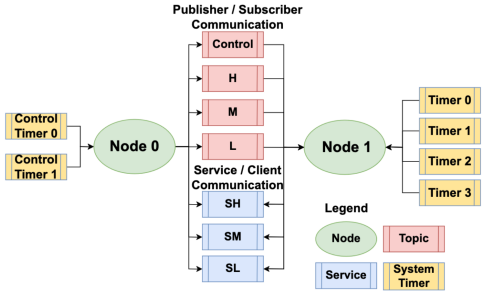


Proof of Concept




```
31
32  # Process
33  events = load_file(path)
34  handler = Ros2Handler.process(events)
35  #handler.data.print_data()
36
37  data_util = Ros2DataModelUtil(handler.data)
38
39  handler.data.print_xml('trace_generated.ros2datamodelmm')
40
```

Proof of Concept



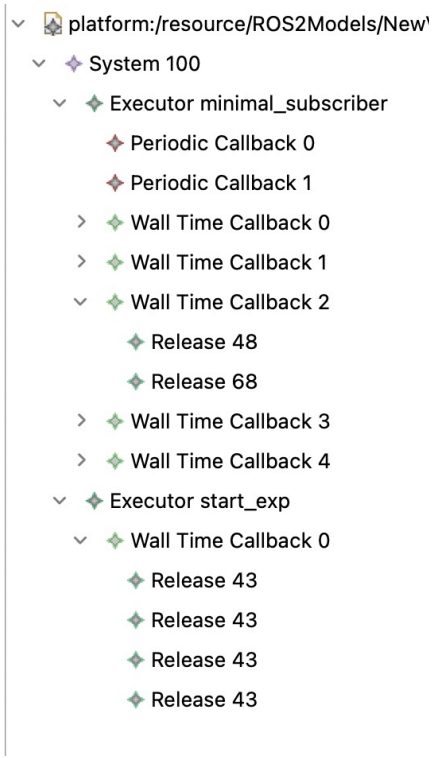
conversionresult

```
====================ROS 2 DATA MODEL====================
Contexts:
                                   timestamp      pid version
context_handle
94072746120384  1677502738930967157  1110802   2.3.0
94435366421696  1677502743279728904  1111072   2.3.0

Nodes:
                                   timestamp      tid      rmw_handle               name namespace
node_handle
94072746126048  1677502738934754734  1110802  94072746280896  minimal_subscriber         /
94435366427344  1677502743282774818  1111072  94435366582272           start_exp         /

Publishers (rmw):
Empty DataFrame
Columns: []
Index: []

Publishers (rcl):
                                   timestamp     node_handle      rmw_handle         topic_name  depth
publisher_handle
140731408565080   1677502738934745454  94072746126048  94072746278384            /rosout   1000
94072746469072    1677502738937134087  94072746126048  94072746541696  /parameter_events   1000
140735833375224   1677502743282766315  94435366427344  94435366579760            /rosout   1000
94435366763072    1677502743285180770  94435366427344  94435366845312  /parameter_events   1000
94435366873392    1677502743286400928  94435366427344  94435366873712             /topic     10
94435366873616    1677502743286575820  94435366427344  94435366930912                /H     10
94435366953648    1677502743286736342  94435366427344  94435366930176                /M     10
94435366968160    1677502743286901099  94435366427344  94435366869568                /L     10

Subscriptions (rmw):
```

- Services /SH
- Services /SM
- Services /SL
- Services /start_exp/get_parameters
- Services /start_exp/get_parameter_types
- Services /start_exp/set_parameters
- Services /start_exp/set_parameters_atomically
- Services /start_exp/describe_parameters
- Services /start_exp/list_parameters
- Publishers /rosout
- Publishers /parameter_events
- Publishers /rosout
- Publishers /parameter_events
- Publishers /topic
- Publishers /H
- Publishers /M
- Publishers /L
- Nodes minimal_subscriber
- Nodes start_exp
- Contexts 9.4072746120384E13
- Contexts 9.4435366421696E13
- Clients /SH
- Clients /SM
- Clients /SL

- platform:/resource/ROS2Models/New
  - System 100
    - Executor minimal_subscriber
      - Periodic Callback 0
      - Periodic Callback 1
      - Wall Time Callback 0
      - Wall Time Callback 1
      - Wall Time Callback 2
        - Release 48
        - Release 68
      - Wall Time Callback 3
      - Wall Time Callback 4
    - Executor start_exp
      - Wall Time Callback 0
        - Release 43
        - Release 43
        - Release 43
        - Release 43

Proof of Concept




```
// 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};
const int releasesSUBSCRIBER1[MAXX]={53,73,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER2[MAXX]={48,68,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER3[MAXX]={43,0,0,0,0,0,0,0,0,0};
const int releasesSUBSCRIBER4[MAXX]={0,0,0,0,0,43,43,43,43,43};

// 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,SUBSCRIBER,10);
SUBSCRIBER1 = WallTimeCallback(1,5,2,releasesSUBSCRIBER1,SUBSCRIBER,10);
SUBSCRIBER2 = WallTimeCallback(2,5,2,releasesSUBSCRIBER2,SUBSCRIBER,10);
SUBSCRIBER3 = WallTimeCallback(3,5,1,releasesSUBSCRIBER3,SUBSCRIBER,10);
SUBSCRIBER4 = WallTimeCallback(4,5,10,releasesSUBSCRIBER4,SUBSCRIBER,1000);

// 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