MARS: a toolset for the safe and secure deployment of heterogeneous distributed systems

Giann Nandi*

David Pereira*

José Proença*

José Santos

Lourenço A. Rodrigues

André Lourenço

Eduardo Tovar*

*CISTER Research Centre

CISTER-TR-221101

2022/12/05

MARS: a toolset for the safe and secure deployment of heterogeneous distributed systems

Giann Nandi*, David Pereira*, José Proença*, José Santos, Lourenço A. Rodrigues, André Lourenço, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: giann@isep.ipp.pt, drp@isep.ipp.pt, pro@isep.ipp.pt, josepedrosantos.2001@gmail.com, lar@cardio-id.com, arl@cardio-id.com, emt@isep.ipp.pt

https://www.cister-labs.pt

Abstract

This work discusses the ongoing development of a toolset named MARS aimed to ease the process of safely deploying runtime verification monitors into distributed micro-ROS and ROS2 nodes. The work is motivated by a use case in the health and automotive domains and covers safety/security concerns around the manipulation of sensitive biometric data.

MARS: a toolset for the safe and secure deployment of heterogeneous distributed systems

Giann Spilere Nandi[1], David Pereira[1], José Proença[1], José Santos[2],
Lourenço A. Rodrigues[2], André Lourenço[2] and Eduardo Tovar[1]

***Abstract*—This work discusses the ongoing development of a toolset named MARS aimed to ease the process of safely deploying runtime verification monitors into distributed micro-ROS and ROS2 nodes. The work is motivated by a use case in the health and automotive domains and covers safety/security concerns around the manipulation of sensitive biometric data.**

***Index Terms*—Safety, Security, Runtime Verification, ROS2, micro-ROS**

I. Introduction

Artificial Intelligence (AI) has been one of the main drivers of innovation in recent years. Its contributions impacted, and continue to impact, several research fields and application domains, including automotive, robotics, and health [16].

Although efforts towards porting AI to extremely resource-constrained devices were made in recent years [1], a large portion of these devices cannot yet process complex AI algorithms in a timely manner locally. A common approach to overcome this limitation is to offload the more resource-intensive tasks to more robust distributed nodes. However, dependable distributed applications may still demand that all network participants comply with safety, security, and real-time requirements.

Developers implementing distributed systems that must comply with such requirements will inevitably face challenges such as how to configure nodes to communicate with other heterogeneous nodes; how to make sure that messages are not disclosed nor manipulated by unauthorized participants (especially in the context of resource-constrained devices); and how to guarantee that data collection and processing is correct and executed in a timely manner.

This work discusses the development of the ***M****onitoring* ***A****rchitecture* ***S****pecification* ***L****anguage* (MARS), a domain-specific language with an associated toolset, introduced on [12], capable of: (i) specifying ROS2-compatible distributed system architectures, (ii) specifying and generating runtime monitors that evaluate a system's correctness while also guaranteeing that their associated overhead will not disrupt the target system's safety requirements, and (iii) generating components that communicate through secure channels. By achieving these goals, we facilitate the process of safely deploying runtime verification monitors into real-time systems, providing a way for teams of engineers with no background in formal methods to deploy runtime monitors in their systems while also providing evidence that these monitors will not disrupt the system's safety with respect to its real-time constraints.

We contextualize and motivate the use of MARS with a use case in the health and automotive domains developed in the context of the VALU3S European R&D project in collaboration with CardioID, a Portuguese company specialized in the analysis and integration of human hearts' biometric data into innovative solutions. The use case builds up on top of a generic and adaptable architecture with a newly developed and publicly available TLS 1.3-based transport that encrypts and authenticates the communication between micro-ROS [17] and ROS2 [18] devices.

II. CardioWheel Use Case

We start by describing our motivating use case provided by CardioID, called *CardioWheel*, used to guide and validate the development efforts of the MARS language and toolset.

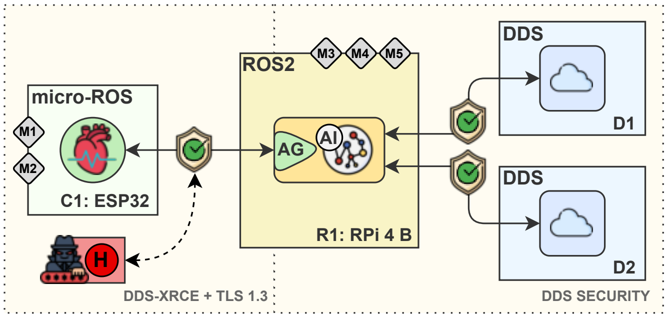

Fig. 1. Node C1 collects, pre-processes, and transmits heart beat signals to R1 over a wireless channel. R1 is analyzes it and forwards results to a set of trusted nodes in the cloud, represented by D1 and D2. The channel between C1 and R1 is secured by our custom transport based on TLS 1.3 and the DDS-XRCE standard.

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UIDP/UIDB/04234/2020); also by FCT within project ECSEL/0016/2019 and from the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey. We would also like to thank Antonio Rodriguez and Pablo Garrido from eProsima for their collaboration in development of the secure custom transport layer of MARS. Disclaimer: This document reflects only the author's view and the Commission is not responsible for any use that may be made of the information it contains)

[1] CISTER Research Centre in Real-Time and Embedded Computing Systems, Polytechnic Institute of Porto, Rua Alfredo Allen, 535, 4200-135, Porto, Portugal. {giann,drp,pro,emt}@isep.ipp.pt

[2] CardioID Technologies Lda, Instituto Superior de Engenharia de Lisboa, Rua Conselheiro Emídio Navarro 1, Room E.06 1959-007 Lisboa, Portugal {jfs, lar, arl}@cardio-id.com

In *CardioWheel*, the heart-beat signals of a vehicle's driver are collected, transmitted wirelessly, and analyzed by AI algorithms. Figure 1 illustrates the system's underlying distributed architecture: a sensor, powered by an Espressif ESP32 microcontroller (C1), is placed in the steering wheel of the vehicle to collect and send aggregated data to a Raspberry PI 4 Model B (R1), which receives it and analyses it using a trained AI model while interacting with remote nodes D1/D2.

While the communication between R1 and D1/D2 complies with the underlying middleware [5] of the full fledged ROS2, C1 uses a stripped-down version called micro-ROS, specially developed to comply with the restrictions of (extremely) resource-constrained devices that have as little memory as 32kB of RAM and 256kB of flash and can run on top of real-time operating systems like FreeRTOS.[1] The information collected by C1 to allows R1 to provide:

- **Biometric Authentication:** by analyzing the user's electrocardiogram (ECG) signals, R1 can authenticate and validate a user's identity analogously to what is done with fingerprints [10].
- **Heart Monitoring:** although unique to every person, ECG signals share common traits that could indicate various body conditions, including heart anomalies, levels of fatigue, and emotional distress [10].

After authenticating the driver, the CardioWheel's inferred levels of fatigue could provide warning signals and potentially adapt parameters during assisted driving. It is important to mention that the quality of the analysis performed by R1 is heavily dependent on the quality of the signals fed to it, requiring C1 to provide low signal-noise readings at high frequency (around 1kHZ). Given the automotive context and the safety hazards that an incorrect system behavior could cause, CardioWheel deploys runtime monitors (M1-M5 in Fig. 1) to check for incorrect system behavior.

Due to the sensitive content of the transmitted data, security and privacy measures are required to protect against cyberattacks. While the communication of R1 with D1 and D2 can be secured by employing the DDS Security protocol [6], no similar option is available to the communication between C1 and R1, leaving the communication channel between C1 and R1 susceptible to attacks coming from a malicious user H, like **(i)** *tampering* with the data transmitted to R1, possibly leading genuine heart problems to be misclassified as a healthy heart behaviour; and **(ii)** *eavesdropping* on sensitive health information, disclosing a user's health condition. We address this lack of security with a newly developed transport, which is discussed in the next section.

III. MARS and Its Associated Toolset

Our work on MARS includes developing a *specification* language and a *back-end*. The specification language compactly and precisely captures both the distributed architecture (including communication channels) and a set of runtime monitors designed to verify safety properties that were too complex to verify statically. The back-end configures the communication channels with the required security levels, generates correct-by-construction runtime monitors, and safely instruments these monitors into a target system

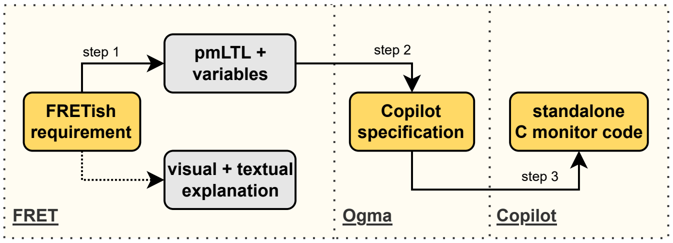

Fig. 2. Workflow adopted by the Ogma tool to synthesize runtime verification monitors from requirements written in the near-natural language of FRETish.

The design of the specification language is ongoing work, partially documented in previous work [12]. This paper focuses on the back-end, i.e., on how to: **(i)** generate runtime monitors from formal specifications, **(ii)** check if the computational overhead caused by these monitors does not disrupt the system's real-time schedulability, and **(iii)** introduce secure communication channels for resource-constrained devices within a micro-ROS/ROS2 distributed environment. Below we address each of these points and clarify the type of systems we support.

Supported Systems Although in the CardioWheel use case we focused on a single application compatible with MARS, we envision our work to easily suit multiple applications as long as they fit into a generic system model. More specifically, we support distributed systems whose nodes comply with the DDS-XRCE [7], the communication middleware for micro-ROS, and DDS [5], the communication middleware for ROS2. Supporting these standards allows excellent flexibility in designing distributed heterogeneous applications while adopting the widespread publish-subscribe message exchange pattern.

With micro-ROS, developers can have (extremely) resource-constrained devices actively interacting with other ROS2 and DDS-compatible nodes in the cloud by publishing and subscribing to topics of interest. On top of that, developers can enforce deterministic and timely behavior on micro-ROS nodes by employing scheduling algorithms to manage the execution of each node's task sets. Finally, MARS focuses on supporting the deployment of software monitors that, at runtime, verify if the system/application complies with a set of formal specifications provided by the user.

Correct-by-Construction Monitor Generation Manually specifying a monitor using formal semantics is error-prone and requires a formal background, usually not present in typical engineering teams. We facilitate the process of specifying and implementing runtime monitors by integrating into MARS tools like Ogma [13] and rmtld3synth [4] that synthesize C/C++ code from specifications written in near-natural languages.

[1] We point the reader to the current list of supported hardware by micro-ROS for further details: https://micro.ros.org/docs/overview/hardware/

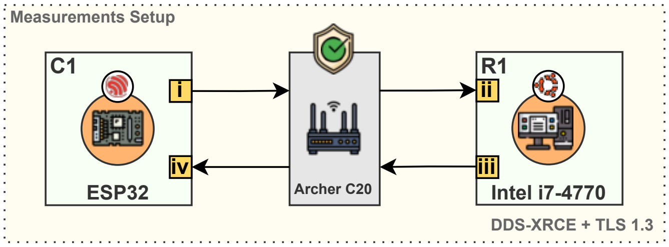

Fig. 3. Setup used to measure the round-trip delay of a message sent by C1. The figure illustrates the hardware used and points to the four additional steps performed by the system when compared to the default transport used by micro-ROS.

To shed some light on the process behind it, we illustrate in Figure 2 the step-by-step workflow of Ogma to transform a specification into a monitor. The workflow starts by receiving as input a requirement written in FRETish [3] describing what needs to be verified in the system and the variables that support such verification. FRET [3] then takes the requirement and variables and provides visual and textual explanations of how the requirement translates to Past-time Metric Linear Temporal Logic (pmLTL) [9]. Ogma then takes the pmLTL formula and associated variables and generates a Copilot [15] monitor specification. Finally, Copilot takes the specification as input and generates the standalone C monitor that MARS will use for its formal analyses and subsequent instrumentation and compilation.

Safe Instrumentation of Runtime Monitors Coupling runtime monitors into a system inevitably incur some computational overhead. In the case of real-time systems, adding a monitor means another task for the real-time operating system to schedule. Depending on its computational impact, adding a monitor could disrupt the system's schedulability and result in unsafe behavior. MARS addresses this concern by employing schedulability analysis algorithms that check the feasibility of scheduling the original task set in addition to the instrumented set of monitors.

Our initial efforts focus on the classical schedulability analysis of simple and static task sets. However, we are also investigating scenarios with systems supporting unbounded mode changes where tasks can be added/removed/have their scheduling parameters modified [2], [8]. For instance, we are currently experimenting with how to verify if a system is schedulable after an unbounded number of mode changes considering the residual accumulative impact that each mode transition can have on the overall schedulability analysis.

Secure Communication Channels MARS addresses the lack of security in the communication between micro-ROS and ROS2 nodes by including in its toolset a newly developed and open-source[2] transport based on TLS 1.3 developed in collaboration with eProsima.[3] The transport allows micro-ROS and ROS2 nodes to authenticate and encrypt their message exchanges with little to no modifications to the system's original behavior. The transport combines the open-source wolfSSL[4] library and the micro-ROS API for custom transports developed by eProsima.

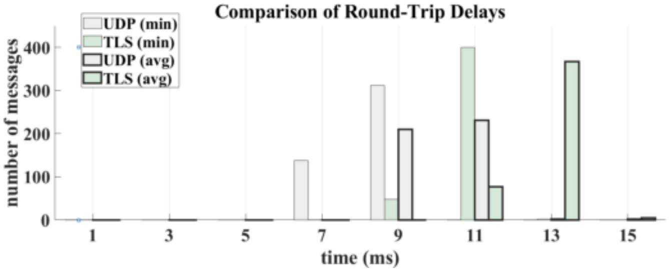

Fig. 4. Round-trip delays measured from 4500 messages sent over 10 identical experiments using the standard micro-ROS transport and 4500 messages sent over 10 identical experiments using our custom TLS 1.3-based transport. The presented histogram shows the minimum and the average round trip delay for each message among all the 10 runs of the experiment for each transport.

Initial benchmarks, illustrated in Figure 4, show an approximate 2-millisecond difference (minimum and average) between the round-trip delay measured of a message sent using the default UDP-based transport of micro-ROS and our transport, which offers the reliability of TCP and the encryption and authentication of TLS 1.3. The hardware setup for the experiments consists of an ESP32, a TPLink router Archer 20, and an Ubuntu Desktop 20.04 equipped with an Intel 4770 processor.

The round-trip delay is the time difference between the timestamp of when C1 receives a message m_i and the timestamp when C1 sends the same m_i message. In total, we ran ten experiments using the UDP transport and ten experiments using our custom transport. In each experiment, we measured the round-trip delay of 450 messages. Figure 4 shows the minimum and the average of each m_i message sent in the network among the ten experiments of each transport layer such that $1 \geq i \leq 450$.

The approximate 2-milliseconds of difference between the two transport layers reflects the overhead caused by the additional reliability guarantees from TCP and the four extra steps, illustrated in Figure 3, that the system needs to perform after the handshake between C1 and R1: i) encryption of m_i by C1; ii) decryption of m_i by R1; iii) encryption of m_i by R1; iv) decryption of m_i by C1.

IV. Related Work and Conclusion

Several works in the literature address the automatic generation of runtime verification monitors from formal specifications. For instance, the work by Perez et al. [14] uses Ogma to generate monitors for ROS2 nodes, and the work of Meredith et al. [11] generates monitors for Java and hardware description languages. However, these approaches do not address the impact of generated software monitors in the context of real-time systems and are incompatible with commercial off-the-

[2] https://bitbucket.org/mars-language/

[3] Developer and maintainer of micro-ROS.

[4] https://www.wolfssl.com/docs/tls13/

shelf microcontrollers and microprocessors, which is precisely the novelty of our work.

By having the harsh resource constraints of these devices in mind, this work presents the current status of the MARS toolset, which is meant to ease the safe instrumentation and deployment of runtime verification monitors in micro-ROS/ROS2/DDS-based distributed systems. More specifically, we discussed the back-end improvements that enable the generation of formally-specified monitors, the employment of formal verification algorithms on the integration of these monitors into real-time systems, and the newly developed secure transport for micro-ROS. The results were motivated and validated by an industrial use case representing the generic system model supported by MARS: a set of micro-ROS nodes with real-time and safety requirements that communicate via publish-subscribe message exchanges that comply with the underlying communication middleware ROS2 and other DDS-based nodes.

References

[1] Francesco Alongi, Nicolo Ghielmetti, Danilo Pau, Federico Terraneo, and William Fornaciari. Tiny neural networks for environmental predictions: An integrated approach with miosix. In *SMARTCOMP 2020*. IEEE.

[2] Hyeongboo Baek, Kang G. Shin, and Jinkyu Lee. Response-time analysis for multi-mode tasks in real-time multiprocessor systems. *IEEE Access*, 8:86111–86129, 2020.

[3] Esther Conrad, Laura Titolo, Dimitra Giannakopoulou, Thomas Pressburger, and Aaron Dutle. A compositional proof framework for fretish requirements. In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2022, page 68–81, New York, NY, USA, 2022. Association for Computing Machinery.

[4] André de Matos Pedro, Jorge Sousa Pinto, David Pereira, and Luís Miguel Pinho. Runtime verification of autopilot systems using a fragment of MTL-$\int$. *International Journal on Software Tools for Technology Transfer*, 20(4):379–395, August 2017.

[5] OBJECT MANAGEMENT GROUP. Data distribution service (dds) specification version 1.4. *http://www.omg.org/spec/DDS/1.4*, 2015.

[6] OBJECT MANAGEMENT GROUP. Dds security specification version 1.1. *https://www.omg.org/spec/DDS-SECURITY/1.1/*, 2018.

[7] OBJECT MANAGEMENT GROUP. Dds for extremely resource constrained environments 1.0. *https://www.omg.org/spec/DDS-XRCE/1.0*, 2020.

[8] Wen-Hung Huang and Jian-Jia Chen. Techniques for schedulability analysis in mode change systems under fixed-priority scheduling. In *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, August 2015.

[9] Martin Leucker and César Sánchez. Regular linear temporal logic. In *Theoretical Aspects of Computing – ICTAC 2007*, pages 291–305. Springer Berlin Heidelberg, 2007.

[10] André Lourenço, Ana Priscila Alves, Carlos Carreiras, Rui Policarpo Duarte, and Ana Fred. CardioWheel: ECG biometrics on the steering wheel. In *Machine Learning and Knowledge Discovery in Databases*, pages 267–270. Springer International Publishing, 2015.

[11] Patrick O'Neil Meredith, Dongyun Jin, Dennis Griffith, Feng Chen, and Grigore Roşu. An overview of the MOP runtime verification framework. *International Journal on Software Tools for Technology Transfer*, 14(3):249–289, April 2011.

[12] Giann Spilere Nandi, David Pereira, José Proença, and Eduardo Tovar. Work-in-progress: a DSL for the safe deployment of runtime monitors in cyber-physical systems. In *RTSS 2020, USA 2020*. IEEE, 2020.

[13] Ivan Perez, Anastasia Mavridou, Tom Pressburger, Alwyn Goodloe, and Dimitra Giannakopoulou. Automated translation of natural language requirements to runtime monitors. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 387–395. Springer International Publishing, 2022.

[14] Ivan Perez, Anastasia Mavridou, Tom Pressburger, Alexander Will, and Patrick J. Martin. Monitoring ros2: from requirements to autonomous robots. 2022.

[15] Lee Pike, Sebastian Niller, and Nis Wegmann. Runtime verification for ultra-critical systems. In *International Conference on Runtime Verification*, pages 310–324. Springer, 2011.

[16] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *ICCUBEA*. IEEE, August 2018.

[17] Jan Staschulat, Ralph Lange, and Dakshina Narahari Dasari. Budget-based real-time executor for micro-ros, 2021.

[18] George Stavrinos. ROS2 for ROS1 users. In *Studies in Computational Intelligence*, pages 31–42. Springer International Publishing, August 2020.