Enhancing compatibility in QoS communication for the Internet of Robotic Things

Flavio Corradini[1], Sara Pettinari[1,*], Barbara Re[1], Luca Ruschioni[1] and Francesco Tiezzi[2]

[1]*School of Science and Technology, University of Camerino, Italy*
[2]*Dipartimento di Statistica, Informatica, Applicazioni "Giuseppe Parenti" (DISIA), University of Florence, Italy*

Abstract
Internet of Things and robotic systems are widespread in many application domains. The emergence of the Internet of Robotic Things seeks to combine both technologies' strengths, thus empowering the system with enhanced capabilities. Nevertheless, this system is composed of heterogeneous devices that need to communicate to work properly. To guarantee effective communication among the devices, we propose a model-driven approach that enables the integration of quality of service policies in communication among Internet of Robotic Things system devices, while ensuring their compatibility at the design time. We apply our approach in a smart agriculture scenario.

Keywords
Internet of Things, Robot, Quality of Service, Model-Driven Development, BPMN

1. Introduction

The Internet of Things (IoT) has seen impressive advancements, enabling cooperation and connection among a wide range of devices to provide smarter services [1]. In parallel, cooperative robotic systems are emerging to accomplish complex tasks in different application domains, e.g., agriculture, manufacturing, and health [2], enhancing the capabilities and services offered by IoT systems. In this context, IoT and robotic systems can be seen as complementary technologies supporting pervasive sensing, tracking and monitoring, and producing action, interaction, and autonomous behavior, respectively [3]. This convergence resulted in the emerging concept of the Internet of Robotic Things (IoRT), which aims to combine the facilities provided by the two [4].

The development of an IoRT system necessitates programming skills that encompass the complexity of both IoT and robotic systems. This also includes managing the message exchanges that facilitate the coordination between these two domains. Specifically, the IoRT comprises interconnected robotic systems interacting with the physical environment through

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal
*Corresponding author.
flavio.corradini@unicam.it (F. Corradini); sara.pettinari@unicam.it (S. Pettinari); barbara.re@unicam.it (B. Re); luca.ruschioni@unicam.it (L. Ruschioni); francesco.tiezzi@unifi.it (F. Tiezzi)
0000-0001-6767-2184 (F. Corradini); 0000-0002-5548-9806 (S. Pettinari); 0000-0001-5374-2364 (B. Re); 0000-0002-7330-4051 (L. Ruschioni); 0000-0003-4740-7521 (F. Tiezzi)

interconnected devices. These interconnections create a dynamic and heterogeneous system where robots can share information, coordinate actions, and provide intelligent services by exploiting IoT devices. Therefore, efficient and reliable communication within the IoRT is crucial to achieving the full potential of these systems. Nevertheless, ensuring devices' communication is trivial due to the heterogeneous nature of devices, protocols, and communication frameworks. Integrating Quality of Service (QoS) policies allows the specification of the communication needs, supporting time-aware, context-aware, and content-aware communications [5], thus impacting the overall system performance. However, while QoS policy integration enhances communication, ensuring effective communication requires devices to integrate these policies in a compatible manner [6].

The objective of this work is to tackle the integration of QoS policies in an IoRT system built on the Robot Operating System (ROS) framework and on the Data Distribution Service (DDS) protocol, with a focus on ensuring communication compatibility among the system's devices. Specifically, ROS is the de facto framework for building robotic applications and relies on the DDS OMG standard to enable communication among distributed devices. Considering these assumptions, we derive the research question that represents the starting point of our approach.

RQ. How can communication compatibility be assessed within a ROS-based IoRT system?

To answer the research question, we propose a model-driven approach that can handle the complexity of representing IoRT systems and guarantee compatibility among QoS policies. Indeed, by exploiting model-driven engineering, it is possible to provide a systematic and structured methodology for designing such systems [7, 8, 9], as well as assess communication compatibility by checking potential issues at design time.

By leveraging this approach, we can develop models that capture the key aspects of the IoRT system. These aspects encompass device interactions, communication protocols, and QoS requirements, all represented through BPMN diagrams. It's worth noting that BPMN is an OMG standard that is expanding its applications both in the robotic [10, 11] and IoT domains [12, 13]. The BPMN application across these domains proves its capability to effectively capture complex devices' behavior and interactions, making it a suitable notation for modeling IoRT systems.

The rest of the paper is structured as follows. Section 2 introduces the core concepts and technologies driving the integration of QoS communication requirements in an IoRT system. Section 3 describes the IoRT case study applied to a smart agricultural scenario and the adopted modeling language. Section 4 presents the model-driven approach for modeling and checking IoRT communication compatibility, along with its application to the case study. Section 5 analyzes the current state-of-the-art for modeling QoS. Finally, Section 6 concludes the paper and discusses future directions.

2. Background

This section describes the core concepts enabling the integration of communication QoS in an IoRT system. We first introduce the ROS framework, explaining its applicability in IoRT scenarios. Then we present the DDS protocol, how it integrates the QoS policies, and how ROS-based systems exploit it.

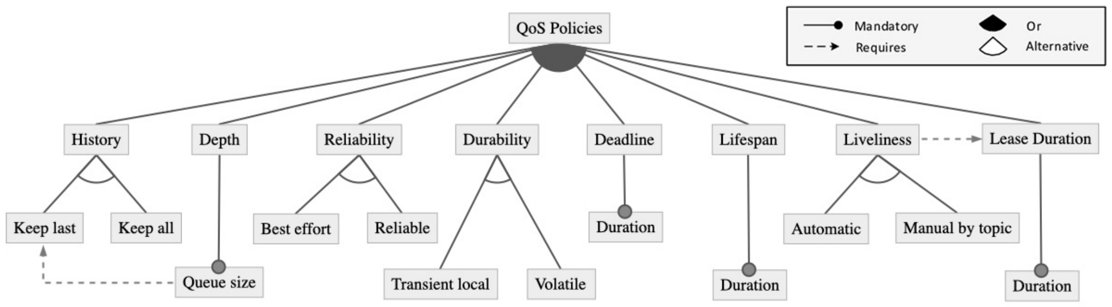


Figure 1: Feature model of ROS QoS policies

2.1. Robot Operating System

ROS[1] is one of the most famous and widely used open-source frameworks for programming robots. It provides an abstraction layer on which developers can build robotics applications. Its second version, i.e., ROS2, has been proposed to achieve full support for multi-robot systems.

ROS2 is designed as a framework of distributed nodes, which are processes able to perform computations and designed to achieve single purposes, e.g., controlling motors or the ultrasonic sensor. Each robot can be seen as a collection of nodes capable of sensing the environment, acting on it, and making decisions. Nodes are in a network and can communicate with each other by sending and receiving data via *topics*. Topics exploit a publish-subscribe pattern allowing to perform topic-based communication, where a message published over a topic can be read by any number of other nodes subscribed to that topic.

Recently, micro-ROS (mROS)[2] has been proposed to foster the integration of IoT devices with ROS-based robots. It provides a basic implementation of the core ROS concepts directly on the device, allowing the deployment of multiple ROS nodes on the same microcontroller [14]. This motivates us to take ROS as a reference framework to implement an IoRT system. For the sake of presentation, we will refer to ROS2 and mROS simply as ROS.

2.2. QoS in Data Distribution Service

The key to enabling the development of multi-interconnected devices in ROS is the OMG standard DDS[3]. This protocol provides low-latency data connectivity, reliability, and scalable architecture, to enhance the development of multi-device applications. Indeed, it enables real-time distributed systems to operate securely as an integrated whole. It introduces a global data space where applications can share information by simply reading and writing data objects [15]. ROS takes advantage of DDS to enable several features. Among them, the one that primarily facilitates the development of IoRT is the distributed discovery system that allows ROS nodes to communicate without using a master node.

[1]https://docs.ros.org/en/rolling
[2]https://micro.ros.org
[3]https://www.omg.org/spec/DDS

Notably, in an IoRT system, communication enables data exchange between devices and thus supports the correct behavior of the system. In this perspective, QoS policies are used to define the communication requirements a system must provide [16]. DDS protocol provides a rich set of QoS policies for controlling data distribution. These policies refer to various communication parameters, such as data availability, resource usage, reliability, and timing [15]. Specifically, ROS implements a subset of the QoS policies provided by DDS protocol, as figured out in the feature diagram in Figure 1. This representation describes the policies considered by ROS with the different values they can assume, as well as the dependencies that may occur between them.

During the development of ROS-based systems, QoS policies can be configured by associating within each publisher and subscriber a QoS profile, i.e., a set of predefined QoS policies or by manually choosing them. However, the manual setup implies the developer must properly configure QoS policies to ensure compatibility. More in detail, the communication between ROS publishers and subscribers is established only if all the QoS policies are compatible, following the compatibility rules defined in the ROS documentation[4]. Notably, multiple subscriptions can be connected to a single publisher simultaneously even if their policies are different. An example of compatibility between QoS policies for the reliability parameter is shown in Table 1.

Table 1
Policy compatibility for the reliability parameter

Publisher	Subscriber	Compatible
Best effort	Best effort	Yes
Best effort	Reliable	No
Reliable	Best effort	Yes
Reliable	Reliable	Yes

3. Case study

The case study we take as a reference is from the smart agriculture domain. We designed a system composed of a weather station, capable of sensing and sharing weather conditions, and a drone operating in the field to monitor and gather data about crop status. Specifically, the drone takes advantage of the collaboration with the weather station to assess the suitability of the weather for executing navigation operations across the agricultural field.

We adopted the BPMN OMG standard notation [17] to enable high-level modeling of the system behavior. Specifically, BPMN is a widely recognized standard that serves various process modeling purposes [18], making it familiar to many users embracing different domains. Its notation, easy to understand and learn, further contributes to its accessibility to users [19]. Additionally, the usage of BPMN, and in particular collaboration diagrams, is motivated by its increasing adoption in model-driven approaches for IoT [13] and robotic systems [11]. Indeed, BPMN collaborations encapsulate in a unique diagram the interplay among control flow, data flow, and communication making the modeling activity easier. The resulting models

[4]https://docs.ros.org/en/rolling/About-Quality-of-Service-Settings.html

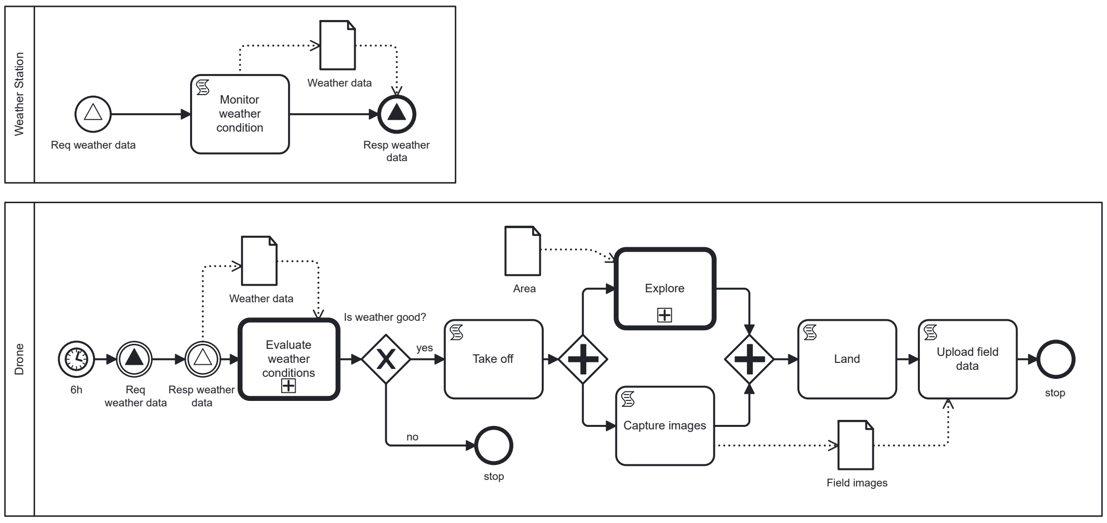


Figure 2: BPMN diagram of smart agriculture case study

are comprehensible to humans and, at the same time, suitable for both model-to-code solutions and direct model execution.

In accordance with the principles presented in [10], which provide comprehensive guidelines for modeling a robotic system in BPMN, we have proposed the BPMN model of the case study.

Figure 2 represents the collaboration diagram of the IoRT behavior. Specifically, publishers and subscribers, defining the communication among the devices, are respectively represented by *throwing* and *catching signals*, simple activities by *script tasks*, complex activities by *call activities*, conditional choices by *XOR gateways* and concurrency by *AND gateways*.

More in detail, the weather station is triggered by a signal catch event when it receives a request for weather data. After that, it computes the requested data by monitoring current weather conditions and sends them with a signal throw event. On the other hand, the drone starts its execution every six hours through a timer start event. It then continues with a request for weather data and waits for a response from the weather station. Once the drone has received the weather data, it evaluates them by enacting the corresponding call activity. If the evaluation results in adverse weather conditions, the drone stops its execution. Differently, if the weather is good, the drone takes off and starts the field exploration while capturing field images. When the exploration activity is finished, the drone executes a landing, uploads the captured data, and ends its execution.

Based on the above-presented case study, we identified the QoS policies, outlined in Table 2, suitable for configuring the publishers and subscribers. The *Req weather data* signal is thrown by the drone asking for the current weather conditions and is caught by the weather station. Whereas, *Resp weather data* is thrown by the weather station to share updated weather conditions and is caught by the drone to evaluate future states. Specifically, all the signals have the *history* parameter associated with the *keep last* policy, which ensures that they store 1 and 5 messages, respectively, as indicated by the *depth* parameter. The request signals have a *reliability* of *best effort*, meaning that the communication attempts to deliver messages but may lose them. In contrast, the response signals require a *reliable* policy to guarantee message delivery. The publisher that responds with weather data is responsible for persisting messages for late-joining subscriptions, indeed the policy associated with the *durability* parameter is *transient local*. In contrast, all the other signals are configured with a *volatile* policy. For all the signals, the maximum expected time interval between successive messages published on a topic is set to 2 seconds, which corresponds to the *deadline* parameter policy. Additionally, the *lifespan* parameter policy is configured to allow a maximum time interval of 10 seconds between the publishing and reception of a message, preventing the message from being considered expired. Finally, the *liveness* of the signals uses the *automatic* policy, ensuring that publishers are considered alive for 10 seconds, i.e., the *lease duration*, after publishing a message.

Table 2
QoS policies for the case study

	Req weather data		**Resp weather data**	
	Pub	**Sub**	**Pub**	**Sub**
History	Keep Last	Keep Last	Keep Last	Keep Last
Depth	1	1	5	5
Reliability	Best Effort	Best Effort	Reliable	Reliable
Durability	Volatile	Volatile	Transient Local	Volatile
Deadline	2	2	2	2
Lifespan	10	10	10	10
Liveliness	Automatic	Automatic	Automatic	Automatic
Lease Duration	10	10	10	10

4. Modeling and checking QoS compatibility

In this section, we introduce our model-driven approach aimed at incorporating compatible QoS policies during the design stage of a ROS-based IoRT system. Initially, we detail our proposed approach and its phases. Afterwards, we introduce the tool which serves as the practical implementation of this approach and aligns with its phases. Lastly, we show the practical application of our approach within the case study presented in the previous section.

4.1. QoS model-driven approach

The model-driven approach we propose, depicted in Figure 3, comprises two specific phases: modeling and QoS compatibility check.

The **modeling phase** is the first step in the design of an IoRT. An IoRT designer leverages BPMN collaboration diagrams to model and capture the behavior of the system. Additionally, within this phase, designers have to select and specify the set of QoS policies that guide device communication.

The modeling of device interactions triggers the QoS policy checking. Indeed, the **QoS compatibility check phase** ensures that the chosen policies are compatible with each other. This phase integrates a checker capable of assessing communication compatibility. It ensures that

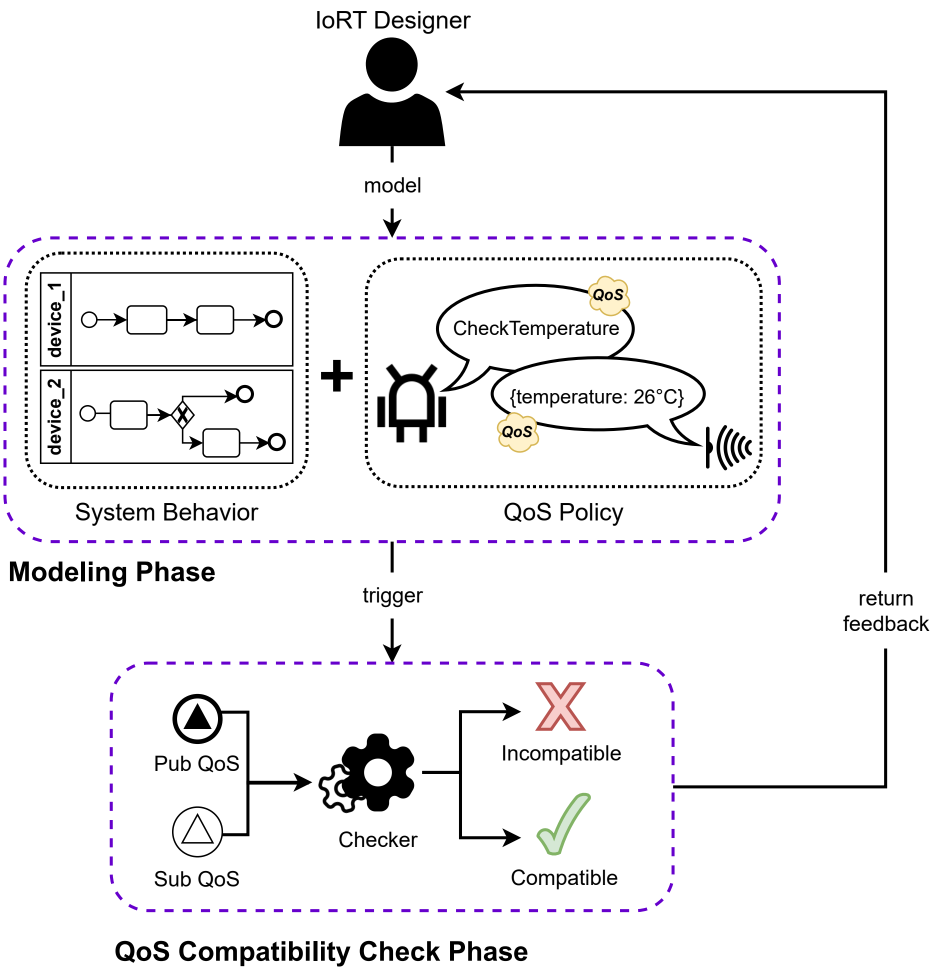


Figure 3: Proposed approach

for each matched publisher and subscriber, the corresponding policies comply with the standards set by ROS for effective communication. The output of this phase is feedback returned to the IoRT designer stating if the checking has been successful or not. Specifically, if incompatible policies have been detected, the designer receives feedback highlighting the need to fix the communication issues in the model.

Summing up, this approach enhances the communication among the various devices that form the IoRT system, contributing to the overall performance of the system. Notably, the BPMN representing the system behavior and enhanced with QoS policies, can be used for guiding the behavior and interactions of the whole system. For instance, it can be translated into ROS-compatible code to be loaded on devices within the IoRT system.

4.2. C-QoS **tool**

To show the applicability of our proposal, we developed the C-QoS tool supporting the approach. In accordance with the approach, C-QoS consists of two main components: a BPMN modeler and a QoS compatibility check module.

The **modeler** has been developed as an extension of the bpmn-js toolkit[5]. It is a web-based interface composed of a palette containing BPMN elements, a central canvas for composing the desired elements into a model, and a property panel for specifying attributes of the elements. The property panel plays a key role in modeling the collaboration features. We extend it to allow the association of a set of QoS policies with each signal node along the modeled behavior. Since BPMN is built upon the XML schema standard, we store the selected policies as new properties associated with the signal definition. Therefore, the values selected during this phase are stored in the diagram, thus enriching the behavioral model with additional information driving devices' communication. An excerpt of the QoS-enhanced BPMN signal is shown in Figure 4.

The **QoS compatibility check** is integrated within our tool using the node-rules[6] module, a rule-engine library that enables real-time control of the inserted QoS policies. Following the ROS-based QoS compatibility specification, we define a specific rule that takes as input each paired publisher and subscriber, along with the respective QoS policies, and assesses their compatibility. An event-based function monitors changes in signal nodes that can occur either when the communication topic changes or when QoS parameter values are modified. Then, the rule engine takes the changed signal node information as input and assesses whether the publisher policy parameters align with the one required by the corresponding subscriber. Specifically, the compatibility is determined based on the *Request vs Offered* model, which allows connections only when the publisher's policies align with the subscriber's policies. Whenever there is a change in the QoS policy of a signal node, the tool triggers the rule engine mechanism to evaluate the updated parameters.

This tool enables the design of the system, identification of communication incompatibilities, and ensures a feedback to the IoRT designer during the system's design phase.

```
<bpmn:intermediateThrowEvent id="Event_0xgazuh"
      qos:history="keep_last" qos:depth="5"
      qos:reliability="reliable" qos:durability="transient_local"
      qos:deadline="2" qos:lifespan="10"
      qos:liveliness="automatic" qos:leaseDuration="10">
      ...
</bpmn:intermediateThrowEvent>
```

Figure 4: QoS-enhanced BPMN signal

[5] https://bpmn.io/
[6] https://github.com/mithunsatheesh/node-rules

4.3. C-QoS at work

Based on the case study presented in Section 3, we illustrate the practical application of the C-QoS tool implementing the proposed approach to model a QoS-enhanced IoRT collaboration diagram.

Exploiting the C-QoS modeling interface, the designer can model the collaboration, that results in the generation of the BPMN diagram represented in Figure 2. Subsequently, the QoS policies can be easily integrated by associating each signal event with the desired set of policies. As shown in Figure 5, by selecting the *Resp weather data* throw signal, the designer can select the desired parameters in the QoS properties panel.

Once all the signals have been modeled or whenever there are modifications to their characteristics, the tool performs compatibility checks. As an illustrative example, we modeled the *Req weather data* signals aiming to generate QoS incompatibilities, specifically concerning the *reliability* and *liveliness* parameters. In detail, we configured the reliability of the publisher with a policy set to *best effort* and the subscriber with a policy set to *reliable.* Whereas the liveliness of the publisher is set to *automatic* and the one of the subscriber is set to *manual by topic.* As stated in the communication standard, these policies are not compatible, thus making communication between the two nodes unfeasible. Figure 6a visually represents the detection of this incompatibility within the model. C-QoS marks the incompatible signals in red and provides, for each involved node, details about which QoS parameters triggered the error, to facilitate the debugging phase for the designer. Differently, we modeled the *Resp weather data* signals in a QoS-compatible way. Figure 6b visually represents a successful modeling, in which the signals are highlighted in green.

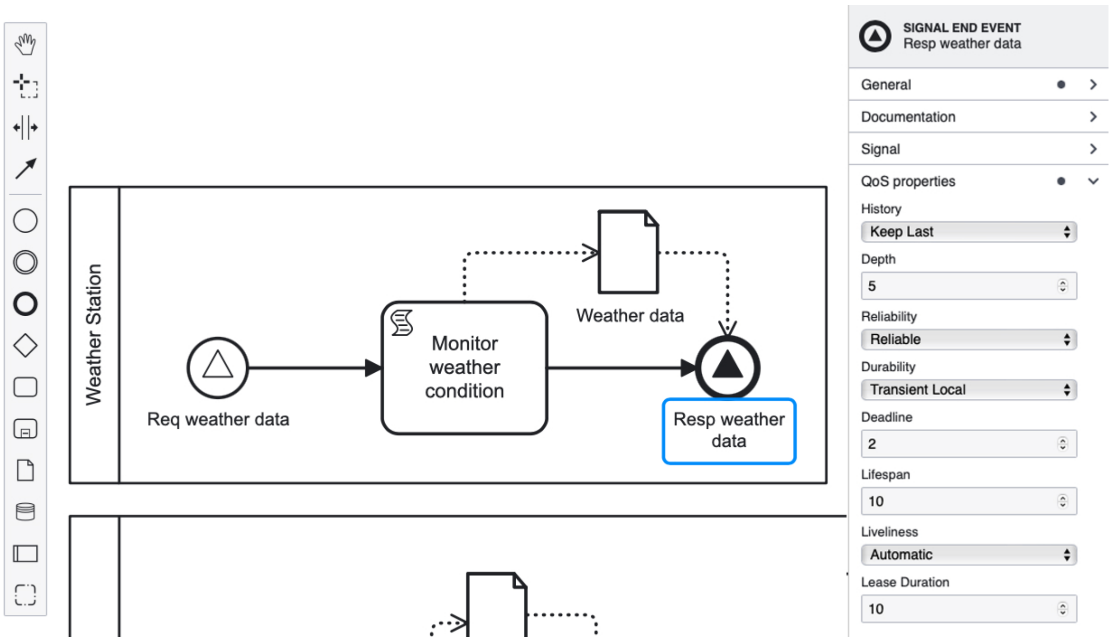
Summing up, the C-QoS tool we propose enables a model-driven solution to specify devices' collaboration in an IoRT system. It guarantees compatibility among the QoS policies as stated in the ROS documentation. Notably, the source code of the tool, as well as an extended example of the smart agriculture scenario are available online[7].

5. Related work

The literature proposes many applications of model-driven approaches targeting QoS policies. Nevertheless, most of them aim to model general QoS in combination with the service level agreement [20, 21], rather than communication QoS metrics. Differently, few works focus on communication policy modeling and their compatibility verification.

Toma et al. [20] highlighted the importance of QoS metrics in service-oriented architecture solutions for realizing distributed applications and ensuring their communication. They proposed the basic steps of modeling QoS characteristics of services with the Web Service Modeling Ontology, providing a QoS-aware service-oriented architecture. In their approach, the Web Service Modeling Language provides support for QoS modeling and attaches QoS characteristics to services and goals. This work differs from our approach in two key aspects. Firstly, it does not integrate a policy-checking mechanism, leaving a gap in ensuring the quality of service in the system. Secondly, it does not provide support to model the system behavior.

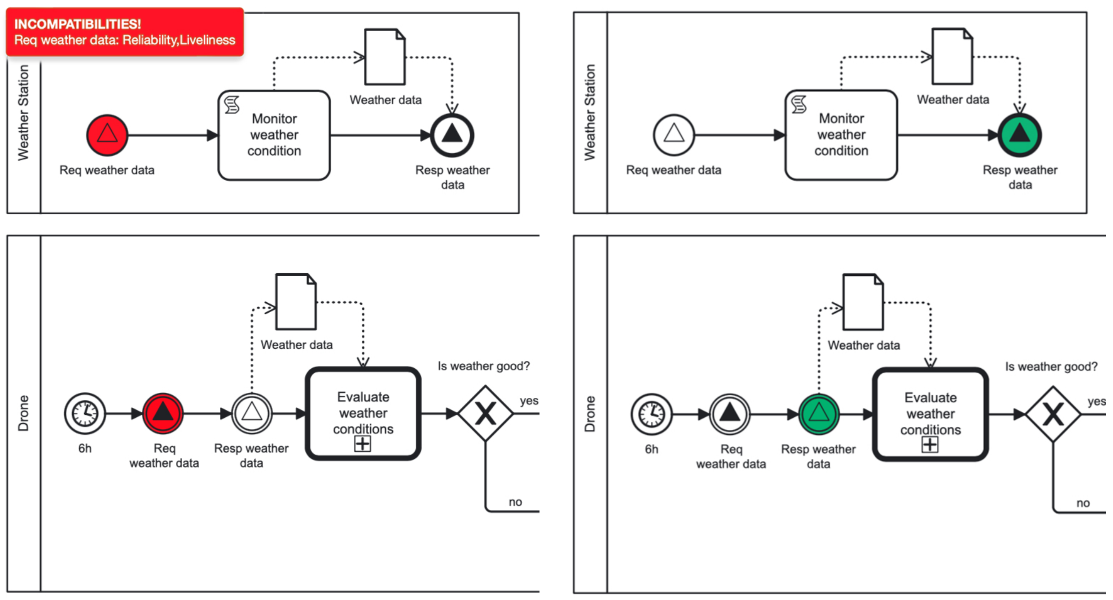
[7]https://pros.unicam.it/c-qos/



History: Keep Last
Depth: 5
Reliability: Reliable
Durability: Transient Local
Deadline: 2
Lifespan: 10
Liveliness: Automatic
Lease Duration: 10

Figure 5: C-QoS: QoS policy modeling

Pérez et al. [22] conducted an analysis of DDS QoS parameters for event-driven applications. The proposed approach represents QoS in distributed real-time systems with end-to-end flow models defined by the MARTE standard. The approach is presented via an automotive case study for advanced driver assistant systems, that require communication reliability, as unexpected behaviors may result in accidents. For modeling system components their solution uses standard profiles but does not consider the inclusion of custom system configurations. Furthermore, the implementation of a compatibility check mechanism is not integrated within the approach. Targeting IoT devices, Archana et al. [23] underlined that a complete analysis of the QoS levels is required to analyze the proper behavior of devices when interacting with each other. The authors aimed to provide an approach to formally model, analyze, and verify the QoS levels of the MQTT protocol. Their solution exploits the PROMELA language and the SPIN model checker tool, to formally describe the desired properties for the communication and verify if they can be satisfied by the MQTT protocol. In contrast with our proposal, this work focuses on formal verification of the MQTT protocol communication, without considering checking QoS levels. Moreover, this approach does not include the modeling of the system behavior. Parra et al. [24] focus on the QoS specification for component-based robotic systems. The authors proposed a domain-specific language to support domain experts in specifying QoS requirements in distributed systems while checking QoS compatibility before executing the system. The proposed solution enables the automatic verification of the QoS requirements at design time, highlighting possible incompatibilities of communication requirements demanded by different parties within the system. The presented approach is applied in a ROS-based robotic system. This approach differs from ours mainly for the adopted modeling language. Notably, using custom domain-specific languages may require time to learn them. Whereas, this effort could



(a) Incompatibility detected

(b) Compatible QoS policies

Figure 6: C-QoS: QoS compatibility check

be mitigated using a well-accepted standard, like the BPMN.

Therefore, differently from the approaches presented in the literature, our approach supports the IoRT developer in modeling the behavior of the system, exploiting the BPMN standard, while ensuring QoS policy compatibility as stated in the DDS standard. The C-QoS tool, in particular, incorporates features designed to help the user in two different ways. On the one hand, it enables the specification of communication QoS requirements during the system design phase of the application. On the other, it performs checks to validate that QoS policies are modeled correctly, thereby guaranteeing communication compatibility among devices within the IoRT system.

6. Conclusions

In this paper, we presented a model-driven approach to handle the complexity of representing IoRT behavior and ensure communication compatibility among devices exploiting QoS policy. The proposed approach allows the IoRT designer to model the behavior of the devices as well as the QoS policy associated with the exchanged messages, using BPMN process models. The produced model is checked to verify the compatibility among the policies thus ensuring the correct behavior of the system. Additionally, the approach is supported by the C-QoS tool, which eases the modeling and the compatibility check of QoS policies.

We demonstrate our approach using a simplified IoRT smart agriculture scenario. Notably, our solution is suitable for handling more intricate use cases, as evidenced on the C-QoS documentation page provided earlier. This flexibility ensures our tool remains robust and relevant in a constantly evolving technological landscape. Furthermore, our tool's adaptability allows it to be customized for systems utilizing other communication protocols that exploit QoS policies. This extension broadens its applicability to a wide range of applications, further enhancing its versatility. Indeed, our approach is not limited just to IoRT systems. It can also be effectively employed in IoT systems and in distributed systems that leverage QoS policies to enable communication among devices.

Exploiting the proposed approach, we can address the research question presented in the introduction. Indeed, the application of a model-driven approach for modeling and checking communication compatibility serves a dual purpose. On the one hand, it simplifies the specification of the system behavior as well as the communication r equirements. On the other, the integration of a checker at design time enables the designer with the capability to assess communication compatibility before system development.

In future work, we plan to validate the approach to better show its effectiveness. This will enable gathering data and insights into the approach's performance and its applicability. Moreover, we plan to integrate this approach within the FaMe framework [11], which enables a direct execution of BPMN process models to drive the execution of autonomous systems. In doing so, we aim to create a comprehensive solution that speeds up the development of IoRT scenarios while associating compatible QoS policies to drive devices' communication.

Acknowledgments

The research underlying this paper was partially supported by the PNRR MUR project ECS_00000041-VITALITY and partially funded by Ministero dell'Università e della Ricerca (MUR), issue D.M. 351/2022 "Borse di Dottorato" - Dottorato di Ricerca di Interesse Nazionale in "Blockchain & Distributed Ledger Technology", under the National Recovery and Resilience Plan (NRRP).

References

[1] R. Gupta, R. Gupta, ABC of Internet of Things: Advancements, benefits, challenges, enablers and facilities of IoT, in: Symposium on Colossal Data Analysis and Networking, IEEE, 2016, pp. 1–5.

[2] Y. Rizk, M. Awad, E. W. Tunstel, Cooperative heterogeneous multi-robot systems: A survey, Computing Surveys 52 (2019) 1–31.

[3] L. A. Grieco, A. Rizzo, S. Colucci, S. Sicari, G. Piro, D. Di Paola, G. Boggia, IoT-aided robotics applications: Technological implications, target domains and open issues, Computer Communications 54 (2014) 32–47.

[4] P. Simoens, M. Dragone, A. Saffiotti, The internet of robotic things: A review of concept, added value and applications, International Journal of Advanced Robotic Systems 15 (2018) 1729881418759424.

[5] O. Vermesan, A. Bröring, E. Tragos, M. Serrano, D. Bacciu, S. Chessa, C. Gallicchio, A. Micheli, M. Dragone, A. Saffiotti, et al., Internet of robotic things–converging sensing/actuating, hyperconnectivity, artificial intelligence and iot platforms, in: Cognitive Hyperconnected Digital Transformation, River Publishers, 2022, pp. 97–155.

[6] P. P. Ray, Internet of robotic things: Concept, technologies, and challenges, IEEE access 4 (2016) 9489–9500.

[7] C. M. Sosa-Reyna, E. Tello-Leal, D. Lara-Alabazares, Methodology for the model-driven development of service oriented iot applications, Journal of Systems Architecture 90 (2018) 15–22.

[8] F. Ciccozzi, R. Spalazzese, Mde4iot: supporting the internet of things with model-driven engineering, in: International Symposium on Intelligent and Distributed Computing, Springer, 2016, pp. 67–76.

[9] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, R. da Silva Barreto, A survey of model driven engineering in robotics, Journal of Computer Languages 62 (2021) 101021.

[10] K. Bourr, F. Corradini, S. Pettinari, B. Re, L. Rossi, F. Tiezzi, Disciplined use of BPMN for mission modeling of Multi-Robot Systems, in: Forum at Practice of Enterprise Modeling, volume 3045, 2021, pp. 1–10.

[11] F. Corradini, S. Pettinari, B. Re, L. Rossi, F. Tiezzi, A BPMN-driven framework for Multi-Robot System development, Robotics and Autonomous Systems 160 (2023) 104322.

[12] Y. Kirikkayis, F. Gallik, M. Winter, M. Reichert, BPMNE4IoT: a framework for modeling, executing and monitoring IoT-driven processes, Future Internet 15 (2023) 90.

[13] P. Valderas, V. Torres, E. Serral, Modelling and executing IoT-enhanced business processes through BPMN and microservices, Journal of Systems and Software 184 (2022) 111139.

[14] K. Belsare, A. C. Rodriguez, P. G. Sánchez, J. Hierro, T. Kołcon, R. Lange, I. Lütkebohle, A. Malki, J. M. Losa, F. Melendez, et al., Micro-ROS, in: Robot Operating System, volume 7, Springer, 2023, pp. 3–55.

[15] OMG, Data Distribution Service (DDS) v. 1.4, 2015.

[16] M. Singh, G. Baranwal, Quality of Service (QoS) in internet of things, in: International Conference On Internet of Things: Smart Innovation and Usages, IEEE, 2018, pp. 1–6.

[17] OMG, Business Process Model and Notation (BPMN) v. 2.0, 2011.

[18] M. Kocbek, G. Jošt, M. Heričko, G. Polančič, Business process model and notation: The current state of affairs, Computer Science and Information Systems 12 (2015) 509–539.

[19] E. Rolón, J. Cardoso, F. García, F. Ruiz, M. Piattini, Analysis and validation of control-flow complexity measures with BPMN process models, in: Workshop on Business Process Modeling, Development and Support, Springer, 2009, pp. 58–70.

[20] I. Toma, D. Foxvog, M. C. Jaeger, Modeling QoS characteristics in WSMO, in: Workshop on Middleware for Service Oriented Computing, 2006, pp. 42–47.

[21] H. Muñoz Frutos, I. Kotsiopoulos, L. M. Vaquero Gonzalez, L. Rodero Merino, Enhancing service selection by semantic QoS, in: The Semantic Web: Research and Applications, Springer, 2009, pp. 565–577.

[22] H. Pérez, J. J. Gutiérrez, Modeling the QoS parameters of DDS for event-driven real-time applications, Journal of Systems and Software 104 (2015) 126–140.

[23] E. Archana, A. Rajeev, A. Kuruvila, R. Narayankutty, J. M. Kannimoola, A formal modeling approach for QoS in MQTT protocol, in: Data Communication and Networks, Springer, 2020, pp. 39–57.

[24] S. Parra, S. Schneider, N. Hochgeschwender, Specifying qos requirements and capabilities for component-based robot software, in: International Workshop on Robotics Software Engineering, IEEE, 2021, pp. 29–36.