Stream Processing for ROS-based Application Development

Alexander Grueneberg
Fakultät Informatik
Hochschule Furtwangen University
Furtwangen, Germany
alexander.grueneberg@hs-furtwangen.de

Alexander Mattes
Fakultät Informatik
Hochschule Furtwangen University
Furtwangen, Germany
al.mattes@hs-furtwangen.de

Lukas Mendel
Fakultät Informatik
Hochschule Furtwangen University
Furtwangen, Germany
lukas.mendel@hs-furtwangen.de

Julian Sobott
Fakultät Informatik
Hochschule Furtwangen University
Furtwangen, Germany
julian.benedikt.sobott@hs-furtwangen.de

***Abstract*—ROS is a middleware platform for integrating different software and hardware components, commonly used in robotics and the automotive sector. Sensors generate continuous data streams that reflect the current state of the system and its environment. While ROS supports message transmission between nodes, it lacks support for data stream processing. Stream processing systems are designed to process data streams, but do not support the integration of different technical components. This work explores how ROS and stream processing systems can be connected to leverage their respective advantages.**

Currently, there is a lack of research into how ROS can be integrated with stream processing systems. We have developed a software platform that maps typed ROS topics to Kafka topics and automatically converts message types. This platform increases development speed and reduces system complexity. Additionally, it greatly facilitates sensor fusion, creating the potential for increased system reliability. A simple use case is employed to validate the platform.

***Index Terms*—ROS 2, Apache Kafka, Apache Kafka Streams, Stream Processing, Complex Event Processing, Sensor Fusion**

I. Introduction

Robot Operating System (ROS) is a middleware platform commonly used in robotics and the automotive sector to connect hardware and software components from different devices. Messages are transmitted between components based on the publish-subscribe pattern [1].

Components such as sensors generate continuous data streams that reflect the current state of the system and its environment. This information can be published via ROS and consumed by ROS components to evaluate the data streams in real time. However, when evaluating the data streams, as is done in Complex Event Processing [2, 3, 4], the limitations of ROS become apparent. ROS lacks sufficient support for the processing of continuous data streams, especially when it comes to aggregations over multiple events (grouping), over multiple streams (joining), or over time (windowing).

Using a streaming system to develop robot applications offers several advantages:

- Sensor fusion: Data from many sensors with overlapping information content may be fused to evaluate the correctness of recorded data [5].
- Development speed: Business processes can be implemented rapidly in streaming systems like Apache Kafka Streams.
- Robot collaboration: By dividing robots into groups and connecting groups across networks via a bridge, communication overhead can be reduced.
- Decoupled development: By separating business logic (modeled in the streaming system) and robot control (modeled in ROS), components can be independently developed by specialized experts.

We developed a software platform to bridge the gap between ROS and Kafka. This platform transforms typed ROS messages into generic Avro messages, which can be transformed into language-specific objects with minimal effort. These generic Avro messages are transmitted using Apache Kafka, a suitable choice as most streaming systems provide Kafka connectors.

When attempting to pass ROS topics to Apache Kafka and process them in another system, the primary challenges include dealing with one-to-many topic mappings and object serialization. It is necessary to have one-to-many topic mappings because on the Kafka side, it is easier to deal with topics, in which all robots publish and subscribe, but on the ROS side every robot typically has its own topics.

We developed several ROS nodes that enforce an architecture that reduces complexity and communication overhead. The registration node encapsulates the logic for adding robots to an existing network, while repeater nodes minimize the need for redundant messages, thus reducing latency and bandwidth usage.

A literature search was conducted to explore existing approaches for combining ROS with streaming systems. Subsequently, we defined a simplified use case for an Industry 4.0

scenario, analyzed general requirements for an autonomous robotic system, and developed a prototype.

In this paper, we present an architecture that captures the findings and challenges, showcasing a possible approach for combining streaming applications with ROS, allowing separate mapping of business logic and robot control.

This paper is structured as follows: In II we present a selection of publications that address bridging between ROS and streaming systems, stream processing and cloud computing in robotics applications, and architectures of ROS-based applications. Chapter III explains the different components and concepts used in ROS-Kafka development. Chapter IV describes a concrete use case for robotics warehouse logistics, extracting a number of requirements for robotics systems. In chapter V, we present the implementation of the chosen scenario before drawing conclusions in chapter VI. Finally, we outline future research directions in chapter VII.

II. Related Works

There are some academic papers that overlap with ours. We categorize them into the following groups: *ROS Bridges*, *Stream Processing*, *Cloud Robotics*, and *Architecture of ROS-based Applications*.

A. ROS Bridges

Given that we are connecting ROS 2 to a streaming system, it is essential to investigate different types of getting data in and out of ROS 2. We have limited ourselves to those works that either provide unique ideas for interoperability or approaches for dealing with the problem of message typing.

Lourenço et al. [6] use Kafka as a broker between ROS 2 and third-party systems (e.g., e-commerce systems in a smart warehouse setting). To achieve this, a pair of ROS 2 nodes is created: 1. the Kafka-ROS bridge node consumes messages from Kafka and publishes them to ROS, and 2. the ROS-Kafka bridge node subscribes to a ROS topic and publishes the messages to Kafka. For QoS profiles with a low depth, they observed message loss in a lower single-digit percentage range that could be addressed by increasing the depth.

RAWFIE is a platform for unmanned vehicles (UxV) [7]. RAWFIE uses Kafka as a distributed message bus. The communication between ROS and Kafka is handled via rosbridge and a Kafka REST proxy. Rosbridge [8] provides a JSON API to ROS functionality for non-ROS programs. A broad range of operations such as subscribing and publishing to topics and calling services are supported. All rosbridge clients communicate with rosbridge over WebSockets. The Kafka REST proxy [9] provides a RESTful interface to a Kafka cluster.

Munro et al. [10] describe a system based on RAWFIE for unmanned aerial vehicles (UAV), but instead of using the rosbridge / Kafka REST proxy bridge as described in [7], they are using a native Kafka bridge implemented in Python, which is capable of dynamically configuring itself using launch files and factory parameters.

Kang et al. [11] describe a bridge between ROS 1 and Open Platform for Robotic Services (OPRoS). OPRoS is a robot software platform including IDE and simulator. The system connects OPRoS components with ROS nodes. One of the key elements of the ROS/OPRoS bridge is the message translation module (MTM) to translate OPRoS messages to ROS messages and vice versa. The MTM manager loads MTM descriptions in a configuration file, and then creates MTM modules.

B. Stream Processing

This category lists papers that address how data can be processed in streams. The focus is on drawing conclusions from robot data.

Heintz et al. [12, 13], describe what stream reasoning is and how it can be used with ROS. The authors built a new framework called DyKnow, which is a bridge between ROS and Kafka. The framework focuses on the problem of dynamic reconfiguration of the application during runtime.

Erich et al. [14] describe how complex event processing can be used as an alternative to the publish-subscribe pattern in robotics applications. They describe the advantages of complex event processing and how graphs can be used to model streams.

Wiener et al. [15] describe how the Apache Stream Pipes framework can be used to build stream processing pipelines. The framework provides adapters for ROS as source and uses Kafka internally. A graphical user interface to build pipelines is provided.

ChoiRbot [16] is a framework whose primary purpose is to orchestrate robots for a common task. The framework works without a central coordination unit and instead relies on direct communication between the robots.

Splash [17] is an extension of ROS 2 that accelerates development speed through model-based software development and code generation. Furthermore, Splash enables stream processing and provides mappings between ROS components and Splash entities to enable stream processing at higher abstraction levels.

C. Cloud Robotics

As indicated in the *ROS Bridges* category, there are many systems that connect a messaging system to a Big Data system [18, 19, 20]. By moving sensor data analysis from the robot to the cloud, it is expected to gain performance and improve scalability in systems with multiple robots. However, high latencies are a problem that can be partially circumvented by edge/fog computing, i.e., by evaluating data closer to its source [21].

D. Architecture of ROS-based Applications

Some papers are concerned with the way ROS systems are built. That is, how nodes are connected, encapsulated, and built to perform complex tasks. These findings are used to inform the system design in the following sections.

Macenski et al. [1] present the individual concepts and components of ROS applications and explain how they differ

between ROS 1 and ROS 2. They show robotic systems that use ROS 2 and provide guidance on how to write and run tests for ROS-based applications.

Reke et al. [22] describe a ROS architecture for autonomous vehicles. They emphasize the importance of creating a separate node for each available sensor and aggregating sensor data using a separate node. In addition, requirements for autonomously moving robots are identified.

Guzman et al. [23] presents a number of projects that use ROS and describes their architectural components. It is shown how these robots can be simulated in ROS' own simulation environment Gazebo, and how application-specific problems were solved.

E. Summary

In summary, there are already concepts that connect ROS with Kafka. We found several papers describing different approaches to extend ROS and thus obtain added value in development or data processing.

Kafka is frequently utilized to facilitate data ingestion into big data systems, like Spark, for batch processing [19, 21, 24, 25, 26]. This is typically a one-way operation, and no data is channeled back to ROS. With the help of stream processing, additional knowledge can be gained. This gained information can be used to improve systems like ROS 2. With the exception of a few works [15, 17, 17, 27], this aspect has hardly been discussed in other works and needs further research.

Lourenço et al. [6] developed a simple ROS bridge that transfers messages as strings between ROS and Kafka. They provide pseudocode, but do not address the complexity of converting arbitrary ROS messages to a common format.

The architecture described by Munro and Clayton [10] contains bridge nodes and Avro serialization supported by schema registries, but it was designed for ROS 1 and there is no source code available. Kang et al. [11] introduce a more abstract Message Translation Module (MTM) instead of a schema registry.

III. Methods

This section provides a description of the architecture and concepts we have developed.

A. Architecture

We have developed a platform that serves as a bridge between ROS 2 and streaming systems. The architecture of this project is illustrated in figure 1 and comprises three components: A ROS 2 application, a streaming application, and a bridge connecting the two. In the ROS 2 application, sensor data is published into ROS topics, and commands from the streaming application are processed. The streaming application receives event streams from the ROS 2 application, processes them, and sends commands back to the ROS 2 application in the form of event streams. To facilitate message exchange between the two systems, a bridge is essential. The bridge's role is to relay messages from one system to the other. Thus, the bridge must translate messages into a suitable format for the receiving side, akin to the approach employed by the Message Translation Module in [11].

For seamless communication between ROS 2 applications and streaming systems, we have opted to utilize a message broker. Employing a message broker simplifies communication between the ROS 2 and streaming applications as the applications do not need to be aware of each other. We have selected Kafka as our message broker due to its extensive support for connectors to streaming systems like Apache Beam, Esper, and Kafka Streams.

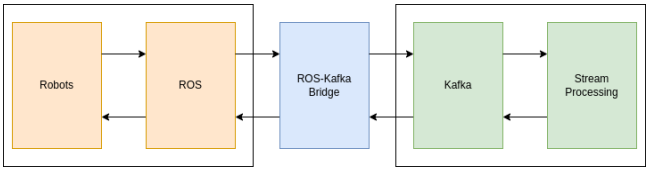

Fig. 1. Integration Architecture

B. ROS-Kafka Bride: roskafka

Roskafka serves as a bridge between ROS 2 and Kafka, with a design similar to the one described in [6]. The /ros_kafka node forwards messages from ROS 2 (via a ROS 2 subscriber) to Kafka (via a Kafka producer), and the /kafka_ros node forwards messages from Kafka (via a Kafka consumer) to ROS 2 (via a ROS 2 publisher). Configurations for the nodes can be set through mappings, which include a mapping name, source topic, destination topic, and message type. Mappings can be added and removed via ROS 2 service calls.

1) Topic Mapping: In ROS 2, multiple robots are typically implemented using namespacing, so each robot has its own set of topics. In stream processing, messages from different instances of the same type are typically collected in a single stream to allow for efficient aggregations. To effectively bridge between both systems, different types of mappings are required: one-to-one, one-to-many, many-to-one, and many-to-many mappings. Many-to-one mappings can be achieved by including the namespace identifier as part of the message (e.g., as metadata). One-to-many mappings can be implemented by using parameterized destination strings in the mapping, where the parameter is extracted from the message.

2) Message Serialization: Both ROS 2 and Kafka require typed messages. Since messages are initially created in ROS 2, we have chosen to base Kafka messages on the ones in ROS 2. Messages are serialized and deserialized using Apache Avro. Avro can be utilized in two ways. In roskafka, we utilize the generic approach, where a dictionary containing the data from the ROS message is passed alongside the schema to a function. This function serializes the data to Avro. The dictionary structure must align with the schema, or else an error will be thrown. The generic approach provides a well-defined interface with relatively little effort. However, in the Kafka Streams application, we prefer specific objects rather than generic ones. Therefore, with the assistance of a plugin, classes are generated for each Avro schema. These classes can

be used in Java as data classes with getters and setters, while also internally handling serialization and deserialization.

To ensure a centralized location for storing these schemas, we employ a schema registry. This approach allows roskafka and the Kafka Streams application to maintain consistent schemas, with any necessary changes only needing to be made in one place.

C. Stream Processing

There are various frameworks available for stream processing, and in this project, we have chosen to utilize Kafka Streams. Kafka Streams offers both a high-level and low-level API, enabling faster development without compromising functionality.

In certain cases, it becomes essential to not only consider the current value of a sensor but also the state of other robots or sensors in order to make complex decisions. To store and access this state information, state stores can be employed. The state is updated with each new value, allowing retrieval of the state of other participants as well.

Since individual sensor values may be prone to errors or inaccuracies, it can be beneficial to aggregate them over a specific time period. Windowing is a technique that can be employed for this purpose. It allows the processing of sensor values within predefined time windows, enabling meaningful analysis and decision-making based on aggregated data.

D. ROS Nodes

In settings with multiple robots, multiple topics have to be mapped for each robot. To simplify this potentially tedious process, we have developed a ROS 2 node that allows for quick and interactive registration of robots of a particular robot type. With this node, topic mappings only need to be defined once, with parameters based on the robot's name. Additionally, the node publishes information about newly registered robots to a topic, enabling potential robot initialization steps to be triggered in nodes that support the registration system.

Streaming systems not only consume continuous streams of events, but may also produce them. However, sending messages multiple times can lead to network congestion and increased processing by the streaming system. To address this, we have developed repetition nodes. These nodes enable messages to be sent only once and then periodically repeated near the consumers of those messages. This approach helps minimize network congestion and reduces processing overhead. Repetition nodes are also useful in dynamic environments where robots may join or leave at any time, ensuring that events are not missed.

E. Availability

roskafka is available at https://gitlab.informatik.hs-furtwangen.de/ss23-forschungsprojekt-7/roskafka
and the ROS nodes are available at https://gitlab.informatik.hs-furtwangen.de/ss23-forschungsprojekt-7/robot_registration.

IV. Use Case and Scenario

A. Concrete Scenario

The use of robotics in industrial and warehouse automation is becoming increasingly prevalent. To showcase the benefits of incorporating streaming systems in robot development, we present the following scenario:

In a warehouse, crates arrive regularly at the goods receiving area. Robots are responsible for picking up these crates and delivering them to their designated storage locations within the warehouse. These robots have the freedom to move independently throughout the facility, allowing for flexible and efficient navigation. However, it is important to note that the warehouse is not solely accessed by robots; human workers, such as maintenance personnel, also frequent the area. Consequently, robots may encounter obstacles or experience hardware failures that prevent them from fulfilling their tasks.

The ultimate objective of deploying robots in this scenario is to minimize human involvement in warehouse operations. To achieve this, the robots should possess the ability to autonomously detect error states, that prevent the system from achieving its goals, and attempt to resolve them without human intervention. If the problem cannot be resolved independently, the robots should provide visual information, such as image recordings, to human operators. This information enables the operators to remotely control the robots or manually address the problem.

B. System Requirements

Based on the above scenario, we can extract a set of system requirements that need to be addressed in the implementation of custom applications and process flows:

- Collaboration on Shared Tasks: The robot fleet must be capable of effectively utilizing shared resources to collectively accomplish tasks. This includes managing physical robots as well as the paths they traverse within the warehouse.
- Fault Detection: In robotic systems, the occurrence of faults or errors cannot be completely eliminated. Factors such as lost network packets or unpredictable environmental influences can impact the robots' performance. To enable appropriate system responses, it is essential to detect deviations from the normal or expected state.
- Automated Problem Resolution: The system should possess the capability to autonomously resolve fault conditions with minimal human intervention. Ideally, this resolution should occur without direct human involvement. However, if human intervention is necessary, the system should provide relevant information and remote control options to expedite problem resolution.
- Decision Support for Human Operators: The system should provide human operators with access to the overall state of the system, enabling them to analyze and address any detected fault conditions. This visibility empowers operators to make informed decisions and take appropriate actions to optimize system performance.

- Dynamic Robot Replacement: The system should facilitate the seamless addition or removal of robots from the active fleet. This flexibility ensures that the system remains functional even when individual robots are taken out of service or new robots are introduced.
- By addressing these system requirements, the integration of streaming systems into robot development for warehouse automation can enhance operational efficiency, fault tolerance, and overall automation capabilities, leading to improved warehouse performance.

V. Results

As a proof of concept, we focused on developing a specific part of the use case, which involved detecting a faulty state of a robot. To carry out the experimentation, we utilized a TurtleBot 4 due to its compatibility with ROS 2 and availability. For the streaming framework, we employed Kafka Streams, as previously discussed.

Kafka Streams supports a pipeline programming model, which simplifies the code structure compared to a sequential approach. A pipeline can consist of multiple transformations such as filter, join, group by, and map, which can be chained. These pipelines can be used for analyzing data from one or many sensors, which is called sensor fusion. Pipelines have the additional advantage, that they support parallelization of tasks.

To detect a faulty state, the data from different sensors were aggregated and analyzed. The relevant topics from the TurtleBot 4 were mapped to Kafka topics using roskafka. The Kafka Streams application subscribed to these topics and performed data analysis within a sliding window. The window allowed us to ensure that a single faulty value did not lead to faulty actions. Within this window, we analyzed if all events indicated a failure of the robot. Only when all events consistently indicated a failure, a command was sent to the robot to change its LED color. The LED color served as an indicator of the robot's state. In the real-world scenario, the other robots would receive a command to take a picture of the failed robot and log relevant information such as position and time.

The architecture of the use case is depicted in figure 2.

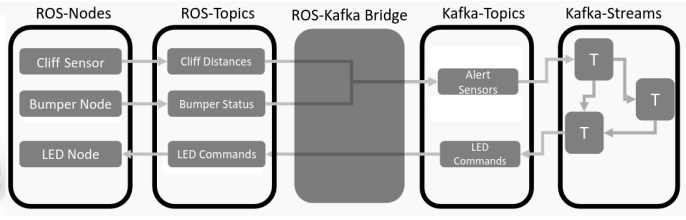

Fig. 2. Architecture of the developed use case

To minimize network traffic, we implemented a mechanism where a single command was sent only when the state of a robot transitioned from running to failure or vice versa. To achieve this, we utilized a state store that stored the current state of each robot. The command was sent only when the detected state differed from the stored state. In cases where a robot required repeated data, the repeater node could be employed.

The code for the hazard detection use case is available at https://gitlab.informatik.hs-furtwangen.de/ss23-forschungsprojekt-7/kafka-streams-turtlebots4.

VI. Conclusion

In this work, we have presented our implementation of a software platform that enables developers to leverage the advantages of stream processing systems when developing ROS applications.

The design of a bridge between ROS and Kafka described by Lourenço et al. in [6] has been improved on by adding dynamic mappings, different mappings types such as one-to-one, one-to-many, many-to-one and many-to-many, and Avro serialization, which enables rapid development on the streaming-side.

By combining ROS 2 with a streaming system, we have identified several advantages for the development of robotics systems.

- Sensor fusion: By combining the data of multiple infrared time-of-flight (TOF) sensors and weight sensors we were able to reliably predict whether a robot was in a faulty state.
- Rapid development: By leveraging the high-level API of Kafka Streams rapid development of streams applications could be achieved, while the usage of its low level API guaranteed no reduction of capability was suffered. The ROS nodes were low in numbers and complexity, since most of the business logic was shifted to Kafka Streams.
- Cooperative robots: While deploying two physical TurtleBot 4 in the same network proved to be impossible due to constraints of the software, we were able to share data between two robots that were deployed in different networks. Another test with simulated robots proved that data from different robots could be aggregated easily.
- Decoupled development: The decoupling of the application into the business logic (Stream Processing) and the robot control (ROS) enabled us to work in separate teams. Definition of an adherence to agreed upon interfaces proved to be simple. This promotes specialization of all team members and increased productivity.

While our platform greatly aids in application development, there are some limitations that result from the server-centric architecture. Many of these problems can be reduced by leveraging pre-aggregation or filtering and the proposed repeater nodes.

- Bottlenecks: A series of potential bottlenecks may present themselves. The physical device running the ROS-Kafka bridge must be powerful enough to deal with the number of messages. Additionally, the central server running the stream processing must be highly available and powerful enough to deal with all incoming requests.
- Network traffic: When many sensors produce data continuously and transmit it via a network, the available

bandwidth will be used up sooner or later. Based on the number of robots within a network, pre-aggregating and filtering data, before it is pushed into Kafka may be advisable.

- Latency: In robotics, time constraints are often critical [28]. By transmitting data over higher distances, the data round trip time is increased. Our proposed stream processing based decision-making should only be employed for non-time-critical decision-making. For example, our platform may check which robots are close enough to potentially crash within the next 10 seconds, and inform them that they should check their relative distance regularly to prevent crashes.

Although we have successfully provided a framework for enhancing ROS development through stream processing systems, further scalability testing is required to assess the platform's performance with a high number of devices and message loads.

We do not advise using stream processing for all tasks. Certain tasks such as navigation, and mapping of the environment are best suited for local execution on ROS nodes.

Our proposed frameworks greatly facilitates multi-robot collaboration by aggregating data from all robots, even across different networks. It represents a valuable tool for improving the capabilities and efficiency of robotics systems.

VII. Future Works

In this section, we present several ideas that can be implemented in the future to further enhance the capabilities of our software platform.

1) Implementation of the complete use case: Due to technical limitations[1] and time constraints, only a portion of the use case was implemented. Future work should focus on implementing the complete use case, which requires at least three robots to be able to collaborate on the same network.
2) Visualization: A web-based user interface (UI) could be developed to visualize the configuration of mappings between ROS topics and Kafka topics. This visualization would be particularly valuable when multiple robots are involved, providing a clear overview of the data flow.
3) Configuration via Web UI: Building upon the visualization aspect, the web UI could be expanded to allow users to configure the mappings between ROS and Kafka topics directly within the interface. This would simplify the configuration process and make it more accessible to users.
4) Monitoring: Building upon the visualization aspect, the web UI could be expanded to allow users to configure the mappings between ROS and Kafka topics directly within the interface. This would simplify the configuration process and make it more accessible to users.
5) Testing: A thorough testing strategy should be developed to ensure the reliability and robustness of the software. This includes defining test cases, performing unit tests, integration tests, and potentially exploring automated testing frameworks for efficient and comprehensive testing coverage.
6) Performance Tests: Conducting performance tests is crucial before deploying the application in a production environment. Key areas to evaluate include round trip time delays, especially for time-critical real-time applications, and scalability with a significant number of messages and robots. Comparisons between serialization formats, such as JSON and Avro, can also be conducted to assess their impact on system performance.
7) Scalability: Investigating scalability aspects and exploring strategies to improve scalability is an important avenue for future research. Intelligent filtering of messages, where not every message is sent to the message broker, can be explored to reduce network traffic and improve overall scalability.

[1] See https://gitlab.informatik.hs-furtwangen.de/ss23-forschungsprojekt-7/documentation/-/tree/main/documentation/#multiple-robots for more information.

References

[1] Steven Macenski et al. "Robot Operating System 2: Design, Architecture, and Uses in the Wild". In: *Science Robotics* 7.66 (May 2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. (Visited on 04/03/2023).

[2] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston: Addison-Wesley, 2002. ISBN: 978-0-201-72789-0.

[3] Juergen Dunkel. "On Complex Event Processing for Sensor Networks". In: *2009 International Symposium on Autonomous Decentralized Systems*. Mar. 2009, pp. 1–6. DOI: 10.1109/ISADS.2009.5207376.

[4] Omran Saleh and Kai-Uwe Sattler. "Distributed Complex Event Processing in Sensor Networks". In: *2013 IEEE 14th International Conference on Mobile Data Management*. Vol. 2. June 2013, pp. 23–26. DOI: 10.1109/MDM.2013.60.

[5] J.A. Castellanos, J. Neira, and J.D. Tardos. "Multisensor Fusion for Simultaneous Localization and Map Building". In: *IEEE Transactions on Robotics and Automation* 17.6 (Dec. 2001), pp. 908–914. ISSN: 2374-958X. DOI: 10.1109/70.976024.

[6] Luan Lucas Lourenço et al. "Achieving Reliable Communication between Kafka and ROS through Bridge Codes". In: *2021 20th International Conference on Advanced Robotics (ICAR)*. Dec. 2021, pp. 324–329. DOI: 10.1109/ICAR53236.2021.9659422.

[7] *D4.1 - High Level Design and Specification of RAWFIE Architecture*. https://www.rawfie.eu/sites/default/files/rawfie_-_d4.1_-_high_level_design_and_specification_of_rawfie_architecture.pdf. (Visited on 04/12/2023).

[8] *Rosbridge_suite - ROS Wiki*. http://wiki.ros.org/rosbridge_suite. (Visited on 04/12/2023).

[9] *Kafka REST Proxy*. Confluent Inc. Apr. 2023. (Visited on 04/12/2023).

[10] Alistair Munro and Gary Clayton. "Drone Swarms, Communications Performance and Big Data". In: *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*. Sept. 2019, pp. 1–5. DOI: 10.1109/VTCFall.2019.8891336.

[11] Jeong Seok Kang, Dong Uk Yu, and Hong Seong Park. "A Robot Software Bridge for Interconnecting OPRoS with ROS". In: *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. Nov. 2012, pp. 296–297. DOI: 10.1109/URAI.2012.6462998.

[12] Daniel de Leng and Fredrik Heintz. "DyKnow: A Dynamically Reconfigurable Stream Reasoning Framework as an Extension to the Robot Operating System". In: *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. Dec. 2016, pp. 55–60. DOI: 10.1109/SIMPAR.2016.7862375.

[13] Daniel de Leng and Fredrik Heintz. "Towards On-Demand Semantic Event Processing for Stream Reasoning". In: *17th International Conference on Information Fusion (FUSION)*. July 2014, pp. 1–8.

[14] Floris Erich and Kenji Suzuki. "Cognitive Robot Programming Using Procedural Parameters and Complex Event Processing". In: *2016 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*. Dec. 2016, pp. 61–66. DOI: 10.1109/SIMPAR.2016.7862376.

[15] Patrick Wiener, Philipp Zehnder, and Dominik Riemer. "Managing Geo-Distributed Stream Processing Pipelines for the IIoT with StreamPipes Edge Extensions". In: *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. DEBS '20. New York, NY, USA: Association for Computing Machinery, July 2020, pp. 165–176. ISBN: 978-1-4503-8028-7. DOI: 10.1145/3401025.3401764. (Visited on 04/03/2023).

[16] Andrea Testa, Andrea Camisa, and Giuseppe Notarstefano. "ChoiRbot: A ROS 2 Toolbox for Cooperative Robotics". In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2714–2720. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2021.3061366. arXiv: 2010.13431 [cs]. (Visited on 04/11/2023).

[17] *Splash on ROS 2: A Runtime Software Framework for Autonomous Machines*. https://ieeexplore.ieee.org/document/9494646. (Visited on 04/03/2023).

[18] Irvin Steve Cardenas, Pradeep Kumar Paladugula, and Jong-Hoon Kim. "Large Scale Distributed Data Processing for a Network of Humanoid Telepresence Robots". In: *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*. Sept. 2020, pp. 1–9. DOI: 10.1109/IEMTRONICS51293.2020.9216366.

[19] Rihab Chaari et al. "Towards a Distributed Computation Offloading Architecture for Cloud Robotics". In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. June 2019, pp. 434–441. DOI: 10.1109/IWCMC.2019.8766504.

[20] Supun Kamburugamuve, Leif Christiansen, and Geoffrey Fox. "A Framework for Real Time Processing of Sensor Data in the Cloud". In: *Journal of Sensors* 2015 (Apr. 2015), e468047. ISSN: 1687-725X. DOI: 10.1155/2015/468047. (Visited on 04/03/2023).

[21] Junwon Lee et al. "Design and Implementation of Edge-Fog-Cloud System through HD Map Generation from LiDAR Data of Autonomous Vehicles". In: *Electronics* 9.12 (Dec. 2020), p. 2084. ISSN: 2079-9292. DOI: 10.3390/electronics9122084. (Visited on 04/03/2023).

[22] Michael Reke et al. "A Self-Driving Car Architecture in ROS2". In: *2020 International SAUPEC/RobMech/PRASA Conference*. Jan. 2020, pp. 1–6. DOI: 10.1109/SAUPEC/RobMech/PRASA48453.2020.9041020.

[23] Roberto Guzmán et al. "Robotnik—Professional Service Robotics Applications with ROS (2)". In: *Robot Operating System (ROS): The Complete Reference (Volume 2)*. Ed. by Anis Koubaa. Studies in Computational Intelligence. Cham: Springer International Publishing, 2017, pp. 419–447. ISBN: 978-3-319-54927-9. DOI: 10.1007/978-3-319-54927-9_13. (Visited on 04/12/2023).

[24] Leigh Duggan et al. "A Rapid Deployment Big Data Computing Platform for Cloud Robotics". In: *International journal of Computer Networks & Communications* 9.6 (Nov. 2017), pp. 77–88. ISSN: 09752293, 09749322. DOI: 10.5121/ijcnc.2017.9606. (Visited on 04/03/2023).

[25] Marc A. Riedlinger et al. "Concept for a Distributed Picking Application Utilizing Robotics and Digital Twins". In: *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. Sept. 2022, pp. 1–4. DOI: 10.1109/ETFA52439.2022.9921659.

[26] ZEKERİYYA DEMİRCİ. *Building A Structured Streaming Data Pipeline*. Mar. 2023. (Visited on 04/03/2023).

[27] Fredrik Heintz. "Semantically Grounded Stream Reasoning Integrated with ROS". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Nov. 2013, pp. 5935–5942. DOI: 10.1109/IROS.2013.6697217.

[28] Christian Lienen and Marco Platzner. *ReconROS Executor: Event-Driven Programming of FPGA-accelerated ROS 2 Applications*. Jan. 2022. DOI: 10.48550/arXiv.2201.07454. arXiv: 2201.07454 [cs]. (Visited on 04/05/2023).