

1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers

**Problem, context, and related work**

Robot Operating System 2 (ROS) is a popular and widely adopted robotics engineering framework. Its main goal is to provide robotics engineers a platform that allows for the development of robotics applications. Framework's infrastructure, which is open source, is based on numerous distributed packages. ROS supports a wide range of devices, starting from small embedded devices up to autonomous industry robots. ROS was developed with real-time capabilities in mind. Since this technology is in its infancy, the ROS community encourages industry and academic research. Academic research in ROS has covered many aspects of robotics development and has achieved quantifiable results regarding soft and hard real-time constraints, lowering latency, and in general, improving performance. However, because of its complicated vertical stack of technologies, there is still room where significant performance benefits could be extracted. The default way to extract performance with aim to lower execution time is by utilizing multi-cores. However, ROS with its complex infrastructure and requirement from robotics engineers to learn parallelization from C/C++ perspective, make it hard to utilize multi-cores properly. Therefore, utilizing multi-cores to lower execution time is one of the main motivators that guides research in ROS. By lowering execution time we gain performance benefits and allow other parts of ROS infrastructure, which are influenced by latency and real-time constraints, to be positively impacted as well. Robots are very heterogeneous devices. Therefore, robots may come with a vast array of different system requirements and system constraints. Some of the system constraints are critical when it comes to the correct functionality of the robots. These critical constraints deal with the overall performance of the robotics application. In the scope of overall performance of the robotics application, the main goal of robotics engineers is to lower execution time or make a particular function execute faster. To facilitate this, we refer to utilization of multi-core processors. However, required specialized knowledge regarding concurrency and expertise, which robotics engineers do not have, slows them down in building their robotics application and decreases efficiency. Robotics engineers are required to understand the architecture of the CPU and how it interacts with main memory. On top of that robotics engineers are required to understand the intricacies regarding caches and data locality. Complemented with this is a software stack that is utilizing the CPU. They need to learn how to use multi-threading libraries, which is not simple. Functions for creating and managing threads require thinking about multiple data-flows in software stack. As a consequence, robotics engineers are required to understand in low-level detail how multi-threading works and how prioritization, scheduling, and affinity combine to solve the problem of lowering execution time. To conclude, the main gap is that robotics engineers do not have an intuitive way of utilizing multi-cores to be able to lower execution time. They are required to know low-level details regarding parallelization and have to develop multi-threading applications. Additionally, robotics engineers do not have accessible interfaces to control Operating System mechanisms regarding prioritization, scheduler type, and affinity

**Requirements**

- SR.01: Framework to facilitate development of multithreaded applications in ROS2
- SR.02: Framework to assist robotics engineers in developing software that takes advantage of multi-cores in order to reduce average execution time, integrated in ROS2.
- SR.02: Pre-locking heap memory - The framework enables pre-locking of heap memory during initialization of the application in order to avoid failed memory allocations, which can occur if application requires significant amount of memory.
- SR.03: Process priority configuration - The framework provides interface for setting priorities of threads in order to avoid priority inversion.
- SR.04: Measurement of heap allocations - The framework offers a visualization on how many memory allocations happened, and it should be easily accessible to the robotics engineers
- SR.05: Measurement of latency. - The framework has a mechanism that

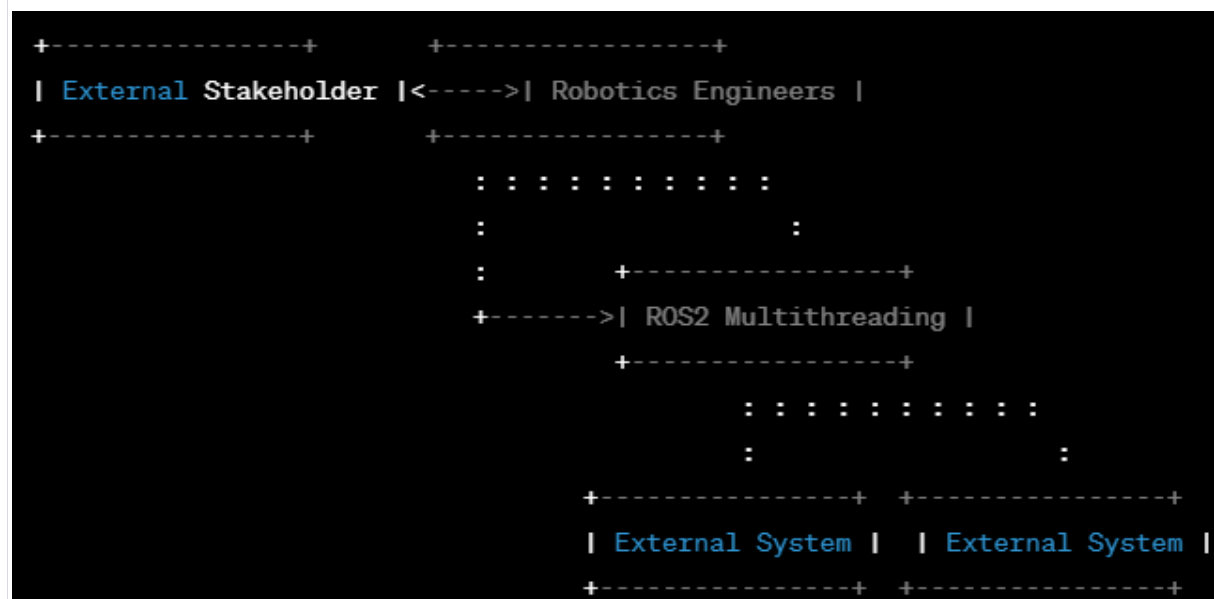
measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: Measurement of memory fetches. - The framework has a mechanism which measures memory fetches. • SR.09: Executor type - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing executor type configuration containing nodes. & ROS executor • SR.10: Node to process assignment - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing configuration. & node assignment, thread, process • SR.11: Allocator type - The framework accepts the message data types that are used throughout the application and it advises robotics engineers on which allocators to use. • SR.12: Easy to use user interface - All information and capabilities of the framework should be gathered in easy-to-use user interface. • SR.13: Guideline - Robotics engineers will be able to express their intentions of parallelizing the code in the framework. The framework will output guidelines for robotics engineers. • SR.14: Parallel Configuration - The framework will accept non parallelized code and will turn it into parallel configuration. & Parallel Configuration Quality requirements • SR.01: Increased CPU Utilization - The framework facilitates the development of multi-threaded applications, and as a result, increased CPU utilization. If the number of cores are  $n$ , (and  $n > 1$ ), then utilization is at least  $0.7 * n$ . • SR.02: Decreased Heap Allocation - The framework facilitates the mechanism with pre-locking memory during the start of the execution of the application. At least 50 percent decrease in heap allocations regarding memory is expected. • SR.03: Increased general performance of the application - The framework facilitates the mechanism for process priority configuration. We expect an increase in general performance in a region of 10 percent. • SR.04: Decreased average execution time - The framework facilitates the mechanism for decreasing average execution time by utilizing multi-threading. • SR.05: Increased CPU Cache utilization - We expect an increase in CPU Cache utilization by 30 percent. • SR.06: Decreased Memory Fetches - We expect to have a decreased memory fetches by 20 percent as a result of increased CPU Cache utilization. • SR.07: Increased productivity of robotics engineers - The facilities of the framework will output parallelized configurations. Compared with by-hand optimization, this process will be expected to take 30 percent less time. & increased productivity • SR.08: Usability - The framework will include an easy to use user interface. We expect robotics engineers to quickly understand the framework and use it efficiently. • SR.09: Modularity - The framework's design will closely follow ROS design philosophy. We expect the framework to be at least modular as ROS and to fit on top of ROS. & modularity Use cases • UC.01: Direct Inverse Kinematics - Sensors, Actuators, Moveable joints, Rotating Joints & Inverse kinematics is a field which concerns itself with correct positioning of the joints in order for end-actuator to reach desired position. As the name indicates, the positioning is worked out from the end-actuator backwards to the first-actuator. • UC.02: Detecting human faces and torso orientation - People in a scene, robot scanning the scene - Image processing is a very important field in robotics engineering. This use case tries to identify the way of doing image processing in robotics, so we can learn how to parallelize common functions. • UC.03: Kalman Filter & A specific robot does sensor fusion and uses kalman filter for prediction - Prediction of the measurements. Kalman filtering is a mathematical approach on how to improve our measures that come from the sensors. • UC.04: Assessing execution time of real-time loop - Real-Time Loops, Function, Time & Since ROS was built with real-time capabilities in mind, measuring real-time loop execution time can give us an insight on performance improvements.

Problem: Robotics engineers using ROS2 struggle with efficiently utilizing multi-core processors to reduce execution time due to the complex infrastructure and required low-level knowledge of concurrency.

Solution: Develop a framework to facilitate the development of multithreaded applications in ROS2 that assists robotics engineers in parallelizing code to reduce execution time. The framework should include features such as pre-locking heap memory, process priority configuration, measurement of heap allocations, latency, jitter, CPU caches usage, and memory fetches, as well as guidance on parallelizing code and outputting parallel configurations. The solution should aim to increase CPU utilization, decrease heap allocation, increase general performance, decrease average execution time, increase CPU cache utilization, and decrease memory fetches, while increasing productivity and usability for robotics engineers. Use cases include direct inverse kinematics, detecting human faces and torso orientation, Kalman filtering, and assessing real-time loop execution time.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (<https://c4model.com/#SystemContextDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Here is a C4 system context diagram for the conceptual solution:



In this diagram, we have two external stakeholders: "External Stakeholder" and "Robotics Engineers." The "External System" is also included in the diagram to represent any external systems that might be relevant to the solution.

The core component of the solution is the "ROS2 Multithreading" framework, which is developed to facilitate the development of multithreaded applications in ROS2. It receives input from the "Robotics Engineers" and outputs parallelized configurations for robotics applications.

The high-level abstract data types transferred between the components and stakeholders include:

- Commands and instructions from "Robotics Engineers" to the "ROS2 Multithreading" framework
- Parallelized configurations and guidance from the "ROS2 Multithreading" framework to "Robotics Engineers"
- Data related to heap memory, process priority, latency, jitter, CPU cache usage, and memory fetches passed between the "ROS2 Multithreading" framework and the external systems

The connections and interfaces between components include the transmission of these data types between "Robotics Engineers" and the "ROS2 Multithreading" framework, as well as the output of parallelized configurations and guidance from the framework to the "Robotics Engineers." Additionally, the "ROS2 Multithreading" framework interacts with the external systems to gather data related to heap memory, process priority, latency, jitter, CPU cache usage, and memory fetches.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (<https://c4model.com/#ContainerDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Here is a C4 container solution diagram for the solution:



The main stakeholder, the Robotics Engineer Client, interacts with the ROS2 Multithreading Server through HTTP/REST API and the ROS2 API. The ROS2 Multithreading Server is responsible for facilitating the development of multithreaded applications in ROS2 by providing recommendations and guidance to the Robotics Engineer Client. The server also utilizes ROS2 Executors and Nodes to optimize the execution of the robotics application.

To address the important quality properties, such as increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, and decreased memory fetches, the following architectural solutions are suggested:

- Architectural style: Client-Server Architecture
- Design Patterns: Facade Pattern, Adapter Pattern
- Design Tactics: Load Balancing, Caching, Resource Pooling
- Components:

- Robotics Engineer Client
- ROS2 Multithreading Server
- ROS2 Executors
- ROS2 Nodes
- Data Storage
- Measurement Tools
- External Systems

The data types transferred between components include:

- ROS2 messages
- Multithreaded application configurations
- Heap memory usage data
- Process priority data
- Latency and jitter data
- CPU cache usage data
- Memory fetches data

The connections and interfaces between components include:

- The Robotics Engineer Client sending ROS2 messages to the ROS2 Multithreading Server through the HTTP/REST API
- The ROS2 Multithreading Server providing guidance and optimized multithreaded application configurations to the Robotics Engineer Client through the HTTP/REST API
- The ROS2 Multithreading Server interacting with ROS2 Executors and Nodes through the ROS2 API
- The ROS2 Multithreading Server utilizing Data Storage and Measurement Tools to gather data related to heap memory, process priority, latency, jitter, CPU cache usage, and memory fetches
- External Systems sending data related to heap memory, process priority, latency, jitter, CPU cache usage, and memory fetches to the ROS2 Multithreading Server

The required and provided interfaces of the components, along with the data types, include:

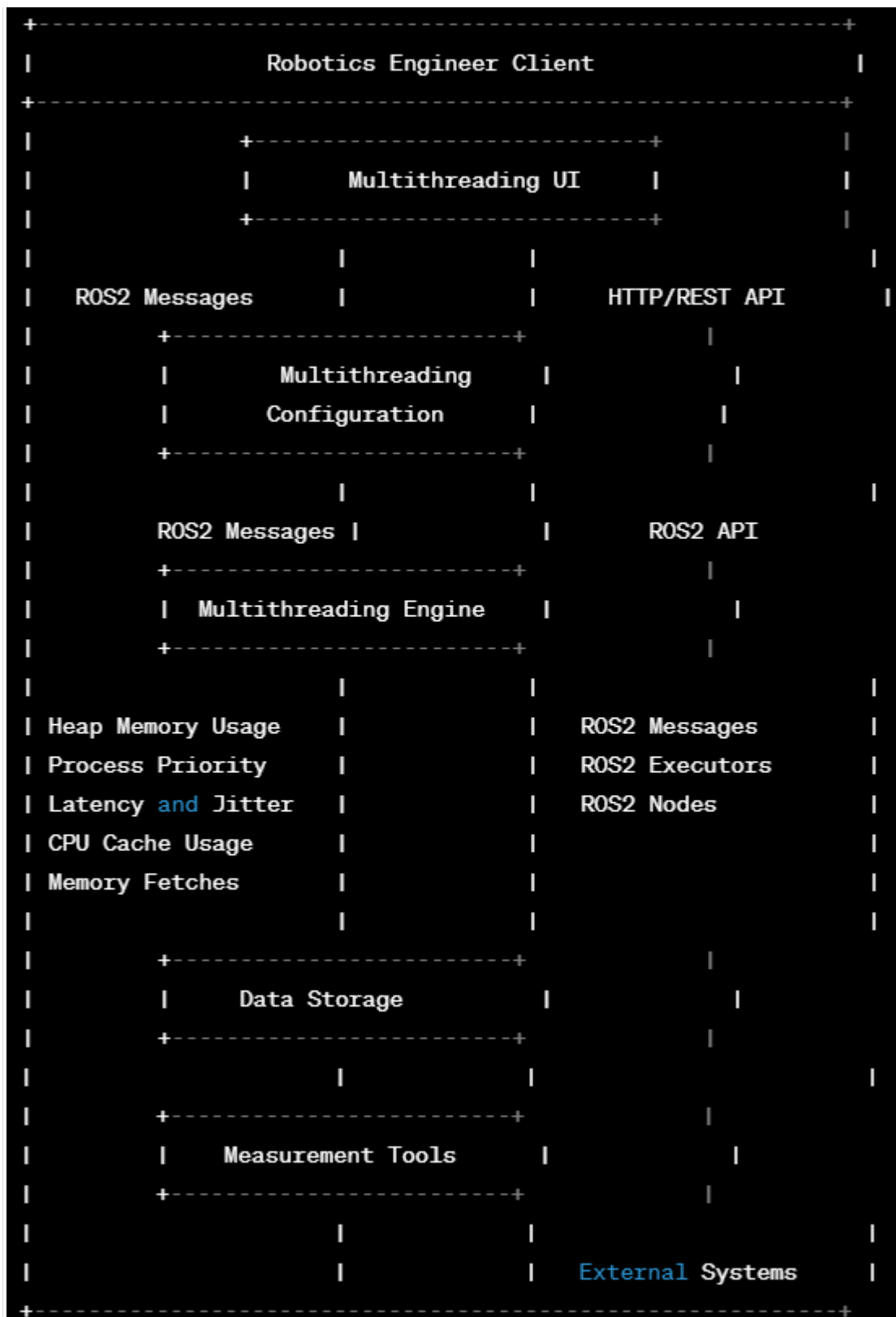
- Robotics Engineer Client:
  - HTTP/REST API: required - ROS2 messages, provided - multithreaded application configurations
- ROS2 Multithreading Server:
  - HTTP/REST API: required - ROS2 messages, provided - multithreaded application configurations
  - ROS2 API: required - ROS2 messages, provided - ROS2 Executors and Nodes
- ROS2 Executors: provided - multithreaded application configurations
- ROS2 Nodes: provided - multithreaded application configurations
- Data Storage: provided - heap memory usage data, process priority data
- Measurement Tools: provided - latency and jitter

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (<https://c4model.com/#ComponentDiagram>). Please use text

(arrows, dots, different shapes, and different colours) to visualise the solution. Consider following:

- Define how the main stakeholder will use the solution. Add it to the diagram.
- Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it.
- Define data types transferred between components.
- Summarise data types transmitted between components. Add this information to the connections and interfaces between components.
- Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types.
- Document which anti-patterns we need to avoid when implementing this solution.

Here is a C4 component solution diagram for the solution:



The Robotics Engineer Client uses the Multithreading UI to interact with the Multithreading Configuration component. The Multithreading Configuration component provides optimized multithreaded application configurations to the Multithreading Engine component, which then interacts with ROS2 Executors and Nodes through the ROS2 API. Data related to heap memory



usage, process priority, latency and jitter, CPU cache usage, and memory fetches are gathered by the Data Storage and Measurement Tools components.

To address the important quality properties, such as increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, and decreased memory fetches, the following architectural solutions are suggested:

- Architectural style: Layered Architecture
- Design Patterns: Facade Pattern, Adapter Pattern
- Design Tactics: Load Balancing, Caching, Resource Pooling
- Components:
  - Robotics Engineer Client
  - Multithreading UI
  - Multithreading Configuration
  - Multithreading Engine
  - ROS2 Executors
  - ROS2 Nodes
  - Data Storage
  - Measurement Tools
  - External Systems

The data types transferred between components include:

- ROS2 messages
- Multithreaded application configurations
- Heap memory usage data
- Process priority data
- Latency and jitter data
- CPU cache usage data
- Memory fetches data

The connections and interfaces between components include:

- The Robotics Engineer Client sending ROS2 messages to the Multithreading Configuration component through the Multithreading UI
- The Multithreading Configuration component providing optimized multithreaded application configurations to the Multithreading Engine component through ROS2 messages
- The Multithreading Engine component interacting with ROS2 Executors and Nodes through the ROS2 API using ROS2 messages
- Data related to heap memory usage, process priority, latency and jitter, CPU cache usage, and memory fetches being gathered by the Data Storage and Measurement Tools components
- External Systems sending data related to heap memory usage, process priority, latency and jitter, CPU cache usage, and memory fetches to the Data Storage and Measurement Tools components through ROS2 messages

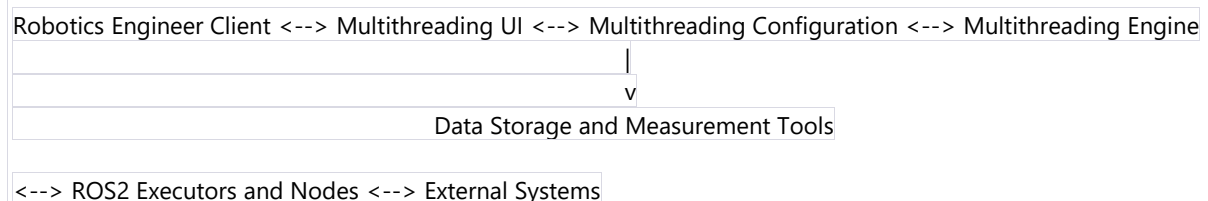
The required and provided interfaces of the components, along

5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

A possible toolchain to implement the solution is:

- ROS2: ROS2 provides the infrastructure for distributed computing and communication between the components.
- C++: C++ is a suitable programming language for high-performance and real-time systems, such as robotics applications.
- Boost: Boost provides a set of C++ libraries, including thread and memory management, which can be used to implement the Multithreading Engine component.
- Qt: Qt is a cross-platform application framework that provides a set of libraries for developing user interfaces, such as the Multithreading UI.

The connections and interfaces between the components are:



The data types transferred between the components include:

- ROS2 messages: used for communication between the Multithreading Configuration component and the Multithreading Engine component, and between the Multithreading Engine component and the ROS2 Executors and Nodes.
- Multithreaded application configurations: used for configuring the Multithreading Engine component.
- Heap memory usage data: used for storing and analyzing heap memory usage statistics.
- Process priority data: used for setting and adjusting process priority levels.
- Latency and jitter data: used for measuring and analyzing latency and jitter.
- CPU cache usage data: used for measuring and analyzing CPU cache usage.
- Memory fetches data: used for measuring and analyzing memory fetches.

The interfaces between the components are:

- Robotics Engineer Client: provides a user interface for interacting with the Multithreading UI component, and sends ROS2 messages to the Multithreading Configuration component.
- Multithreading UI: provides a user interface for configuring multithreaded applications, and sends multithreaded application configurations to the Multithreading Configuration component.

- Multithreading Configuration: receives multithreaded application configurations from the Multithreading UI component, and sends ROS2 messages to the Multithreading Engine component.
- Multithreading Engine: receives ROS2 messages from the Multithreading Configuration component, and interacts with ROS2 Executors and Nodes through the ROS2 API using ROS2 messages. Sends data related to heap memory usage, process priority, latency and jitter, CPU cache usage, and memory fetches to the Data Storage and Measurement Tools components.
- ROS2 Executors and Nodes: receive ROS2 messages from the Multithreading Engine component, and perform the specified tasks.
- External Systems: send data related to heap memory usage, process priority, latency and jitter, CPU cache usage, and memory fetches to the Data Storage and Measurement Tools components through ROS2 messages.
- Data Storage and Measurement Tools: receive data related to heap memory usage, process priority, latency and jitter, CPU cache usage, and memory fetches from the Multithreading Engine component and the External Systems through ROS2 messages, and store and analyze the data.