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 for the solution Functional requirements • SR.01: Framework to facilitate development of multithreaded applications in ROS2 - 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

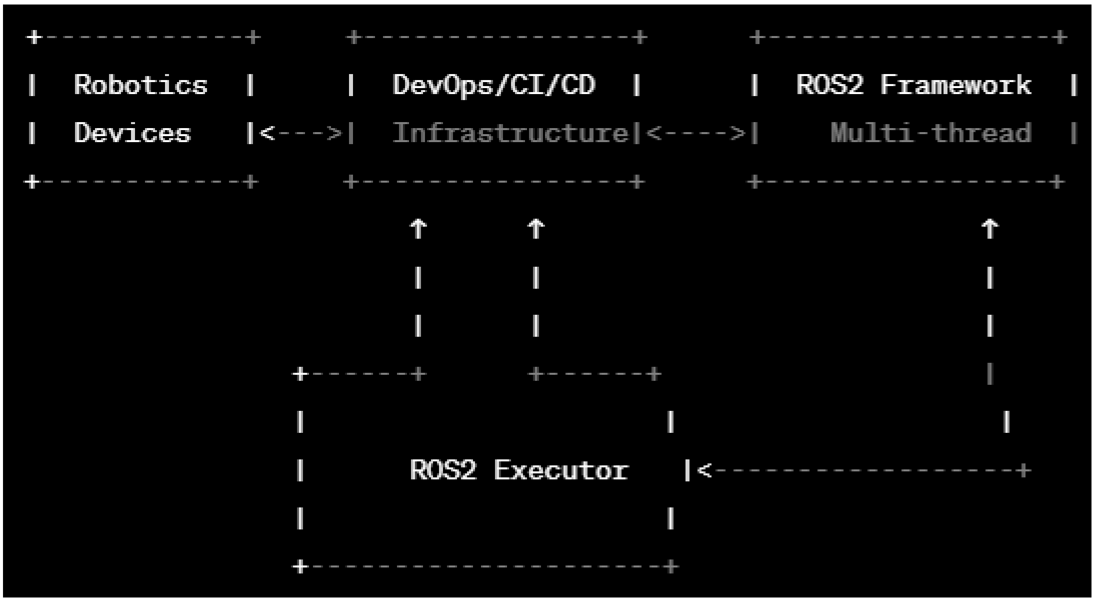
the robotics engineers • SR.05: Measurement of latency. - The framework has a mechanism that measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: 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 the Robot Operating System 2 (ROS) have difficulty utilizing multi-core processors to lower execution time and improve performance due to the complex infrastructure and requirement to learn parallelization from a low-level perspective.

Solution: A software system architecture solution that includes a framework to facilitate the development of multi-threaded applications in ROS2, integrated with the ROS executor, process priority configuration, pre-locking heap memory during initialization, measurement of heap allocations, latency, jitter, CPU cache usage, and memory fetches. The framework should also provide node to process assignment, allocator type advice, easy-to-use user interface, and output parallelized configurations. Quality requirements include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. Use cases include direct inverse kinematics, detecting human faces and torso orientation, Kalman filter, and assessing execution time of real-time loops.

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.

Please find below the C4 system context diagram for the conceptual solution:



```
+-----------+     +---------------+      +----------------+
| Robotics  |     | DevOps/CI/CD  |      | ROS2 Framework |
| Devices   |<--->| Infrastructure|<---->|   Multi-thread |
+-----------+     +---------------+      +----------------+
                      ↑      ↑                      ↑
                      |      |                      |
                      |      |                      |
                +-----+      +------+               |
                |                   |                |
                |     ROS2 Executor  |<----------------+
                |                   |
                +-------------------+
```

External Systems/Stakeholders:

- Robotics Devices: The physical robots that run the robotics applications.
- DevOps/CI/CD Infrastructure: The infrastructure that enables Continuous Integration and Continuous Deployment of robotics applications.
- Robotics Engineers: The stakeholders responsible for developing the robotics applications.

Abstract Data Types:

- ROS2 messages: Data types transmitted between the robotics devices and the ROS2 Framework.

- Multi-threading messages: Data types transmitted between the ROS2 Framework and the executor for multi-threading.

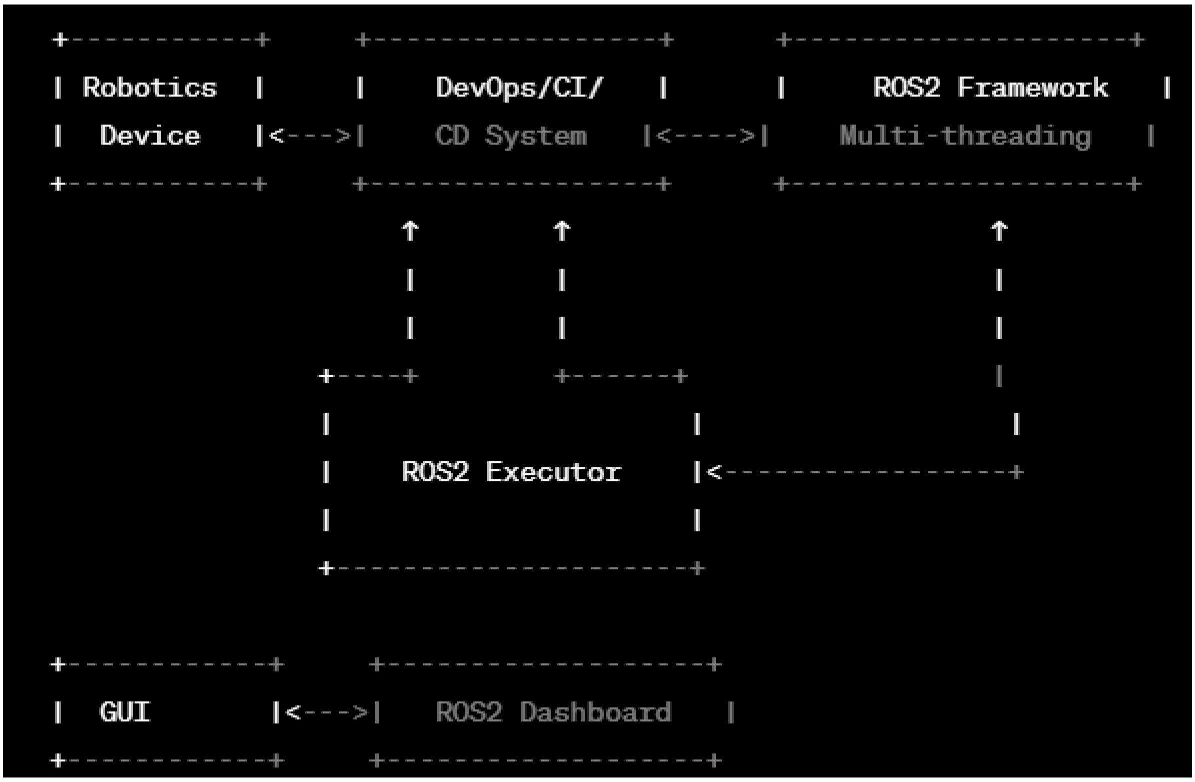
Connections/Interfaces:

- Robotics Devices send ROS2 messages to the ROS2 Framework to execute the robotics application.
- The ROS2 Framework utilizes multi-threading to process the ROS2 messages and sends multi-threading messages to the ROS2 executor.
- The ROS2 executor executes the multi-threading messages to lower the execution time of the robotics application.
- The DevOps/CI/CD infrastructure deploys the robotics application to the Robotics Devices.

The proposed architecture consists of three main components: Robotics Devices, DevOps/CI/CD Infrastructure, and the ROS2 Framework. The Robotics Devices interact with the ROS2 Framework by sending ROS2 messages to execute the robotics application. The ROS2 Framework utilizes multi-threading to process the ROS2 messages and improve performance. It then sends multi-threading messages to the ROS2 executor for execution. The DevOps/CI/CD infrastructure is responsible for deploying the robotics application to the Robotics Devices.

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.

Please find below the C4 container solution diagram for the proposed architecture:



```
+-----------+     +-----------------+      +--------------------+
| Robotics  |     |    DevOps/CI/   |      |    ROS2 Framework  |
|  Device   |<--->|    CD System    |<---->|   Multi-threading  |
+-----------+     +-----------------+      +--------------------+
                     ↑        ↑                         ↑
                     |        |                         |
                     |        |                         |
                +----+        +------+                  |
                |                     |                  |
                |    ROS2 Executor    |<-----------------+
                |                     |
                +---------------------+


+------------+     +-------------------+
|  GUI       |<--->|   ROS2 Dashboard   |
+------------+     +-------------------+
```

Stakeholder Usage:

- Robotics Engineers will use the ROS2 Framework with the help of DevOps/CI/CD system to develop and deploy robotics applications on Robotics Devices. They will use the ROS2 Dashboard to monitor the performance of the applications.

Quality Properties:

- Performance: The framework should utilize multi-threading to improve performance.
- Usability: The framework should have an easy-to-use interface for robotics engineers.
- Maintainability: The framework should be modular to allow for easy maintenance and scalability.

Architectural Components:

- Robotics Devices: The physical robots that run the robotics applications.
- DevOps/CI/CD System: The infrastructure that enables Continuous Integration and Continuous Deployment of robotics applications.
- ROS2 Framework: The framework that facilitates the development of multi-threaded applications in ROS2.
- ROS2 Executor: The component responsible for executing multi-threaded messages in the ROS2 Framework.
- GUI: The graphical user interface that allows robotics engineers to interact with the ROS2 Framework.
- ROS2 Dashboard: The dashboard that provides monitoring and visualization of the performance of the robotics applications.

Data Types:

- ROS2 messages: Data types transferred between Robotics Devices and the ROS2 Framework.
- Multi-threading messages: Data types transferred between the ROS2 Framework and the ROS2 Executor.
- Dashboard messages: Data types transferred between the ROS2 Framework and the ROS2 Dashboard.

Interfaces:

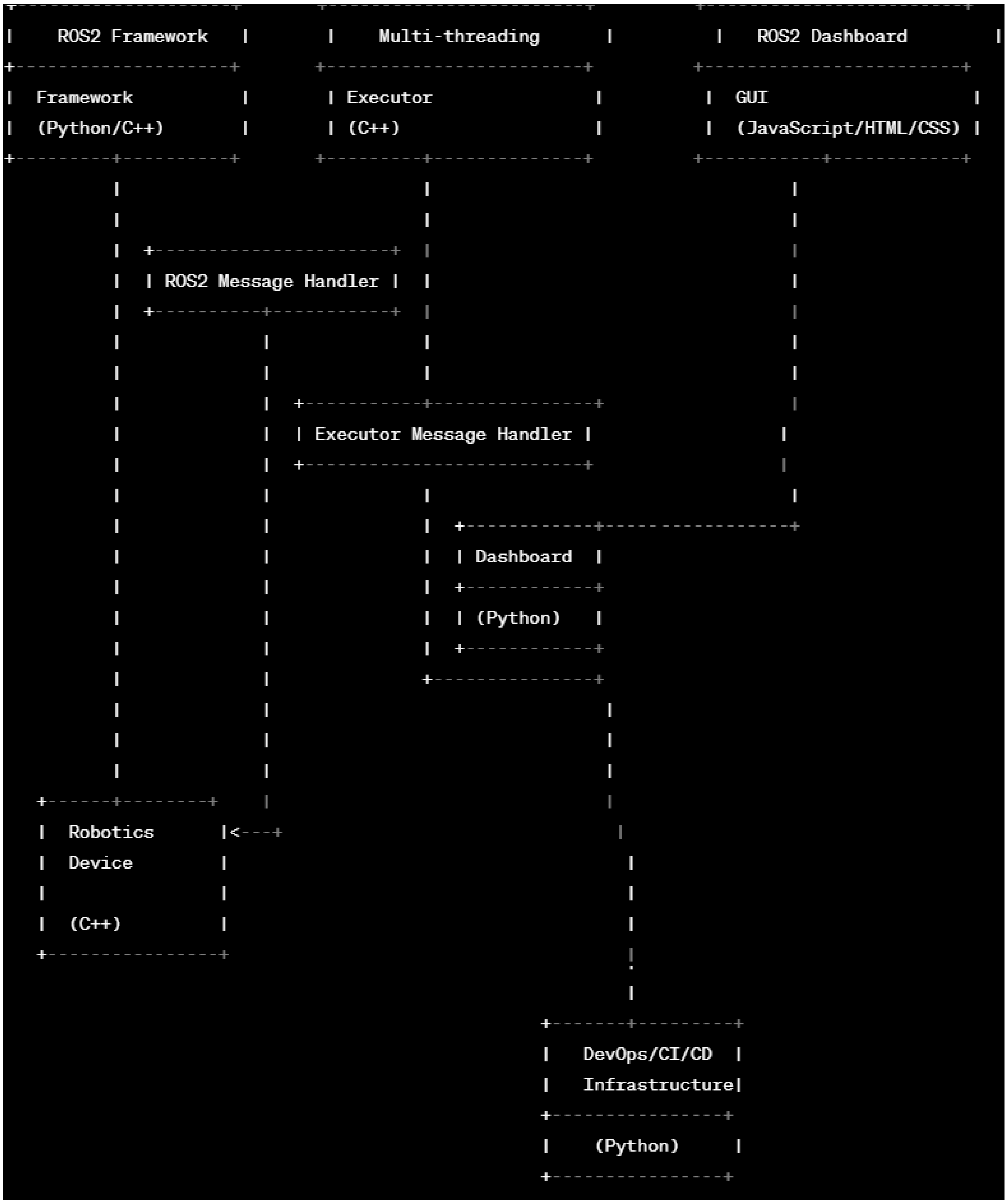
- Robotics Devices send ROS2 messages to the ROS2 Framework.
- The ROS2 Framework utilizes multi-threading to process the ROS2 messages and sends multi-threading messages to the ROS2 executor.
- The GUI provides an interface for robotics engineers to interact with the ROS2 Framework.
- The ROS2 Framework sends dashboard messages to the ROS2 Dashboard for monitoring and visualization.

Anti-patterns:

- Monolithic Architecture: The solution should avoid creating a monolithic architecture by keeping the components modular and independent.
- Over-engineering: The solution should avoid over-engineering by focusing on the requirements and needs of the stakeholders.

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.

Please find below the C4 component solution diagram for the proposed architecture:



```
+---------------------+      +-------------------------+        +-------------------------+
|    ROS2 Framework   |       |    Multi-threading      |          |  ROS2 Dashboard         |
+---------------------+      +-------------------------+        +-------------------------+
|  Framework          |       | Executor               |          | GUI                   |
|  (Python/C++)       |       | (C++)                  |          | (JavaScript/HTML/CSS) |
+---------+-----------+      +---------+---------------+        +------------+------------+
          |                            |                                  |
          |                            |                                  |
          |  +-----------------------+ |                                  |
          |  | ROS2 Message Handler  | |                                  |
          |  +----------+------------+ |                                  |
          |             |              |                                  |
          |             |              |                                  |
          |             |  +-----------+--------------+                   |
          |             |  | Executor Message Handler |                  |
          |             |  +--------------------------+                  |
          |             |              |                                  |
          |             |              |  +------------+------------------+
          |             |              |  | Dashboard  |
          |             |              |  +------------+
          |             |              |  | (Python)   |
          |             |              |  +------------+
          |             |              +---------------+
          |             |                               |
          |             |                               |
          |             |                               |
   +------+---------+   |                               |
   |  Robotics     |<---+                               |
   |  Device       |                                     |
   |               |                                     |
   |  (C++)        |                                     |
   +----------------+                                     |
                                                          |
                                                  +-------+---------+
                                                  |   DevOps/CI/CD  |
                                                  |   Infrastructure|
                                                  +----------------+
                                                  |    (Python)     |
                                                  +----------------+
```

Stakeholder Usage:

- Robotics Engineers will use the ROS2 Framework with the help of DevOps/CI/CD system to develop and deploy robotics applications on Robotics Devices. They will use the ROS2 Dashboard to monitor the performance of the applications.

Quality Properties:

- Performance: The framework should utilize multi-threading to improve performance.

- Usability: The framework should have an easy-to-use interface for robotics engineers.
- Maintainability: The framework should be modular to allow for easy maintenance and scalability.

Architectural Components:

- ROS2 Framework: The framework that facilitates the development of multi-threaded applications in ROS2.
- Multi-threading Executor: The component responsible for executing multi-threaded messages in the ROS2 Framework.
- ROS2 Message Handler: The component responsible for handling ROS2 messages.
- Executor Message Handler: The component responsible for handling multi-threading messages.
- ROS2 Dashboard: The dashboard that provides monitoring and visualization of the performance of the robotics applications.
- GUI: The graphical user interface that allows robotics engineers to interact with the ROS2 Framework.
- DevOps/CI/CD Infrastructure: The infrastructure that enables Continuous Integration and Continuous Deployment of robotics applications.
- Robotics Devices: The physical robots that run the robotics applications.

Data Types:

- ROS2 messages: Data types transferred between Robotics Devices and the ROS2 Framework.
- Multi-threading messages: Data types transferred between the ROS2 Framework and the Multi-threading Executor.
- Dashboard messages: Data types transferred between the ROS2 Framework and the ROS2 Dashboard.

Interfaces:

- ROS2 Message Handler provides an interface to the ROS2 Framework for handling ROS2 messages.
- Executor Message Handler provides an interface to the Multi-threading Executor for handling multi-threading messages.
- The ROS2

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.

For the proposed architecture, we can use a variety of tools and technologies to implement the solution. Here's a suggested toolchain:

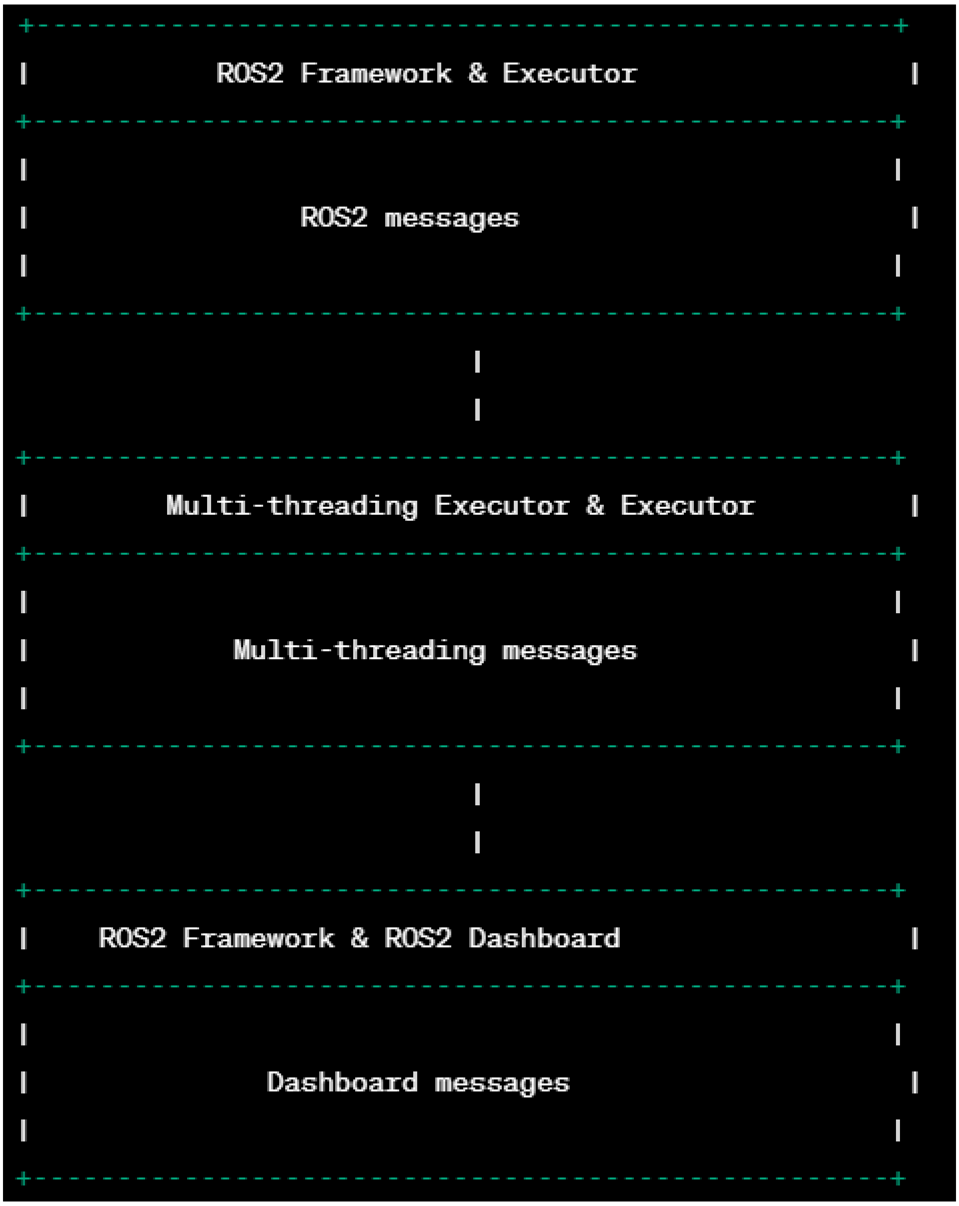
- ROS2 Framework: The ROS2 Framework is already an existing open-source framework and can be used as is.

- Multi-threading Executor: For the multi-threading executor, we can use the C++ standard thread library.
- ROS2 Message Handler: For the ROS2 Message Handler, we can use the ROS2 messaging library.
- Executor Message Handler: For the Executor Message Handler, we can use the Boost C++ library.
- ROS2 Dashboard: For the ROS2 Dashboard, we can use the web framework Flask with Python for the backend and ReactJS for the frontend.
- GUI: For the GUI, we can use PyQt5 with Python.
- DevOps/CI/CD Infrastructure: For the DevOps/CI/CD infrastructure, we can use tools like Jenkins, GitLab, Docker, and Kubernetes.
- Robotics Devices: For the Robotics Devices, we can use various hardware platforms such as Raspberry Pi, NVIDIA Jetson, or other embedded systems.

Data Types:

- ROS2 messages: These are the data types transferred between Robotics Devices and the ROS2 Framework. They are defined using ROS2 message definition files (.msg files).
- Multi-threading messages: These are the data types transferred between the ROS2 Framework and the Multi-threading Executor. They are defined using C++ structs or classes.
- Dashboard messages: These are the data types transferred between the ROS2 Framework and the ROS2 Dashboard. They are defined using JSON or other web-compatible data formats.

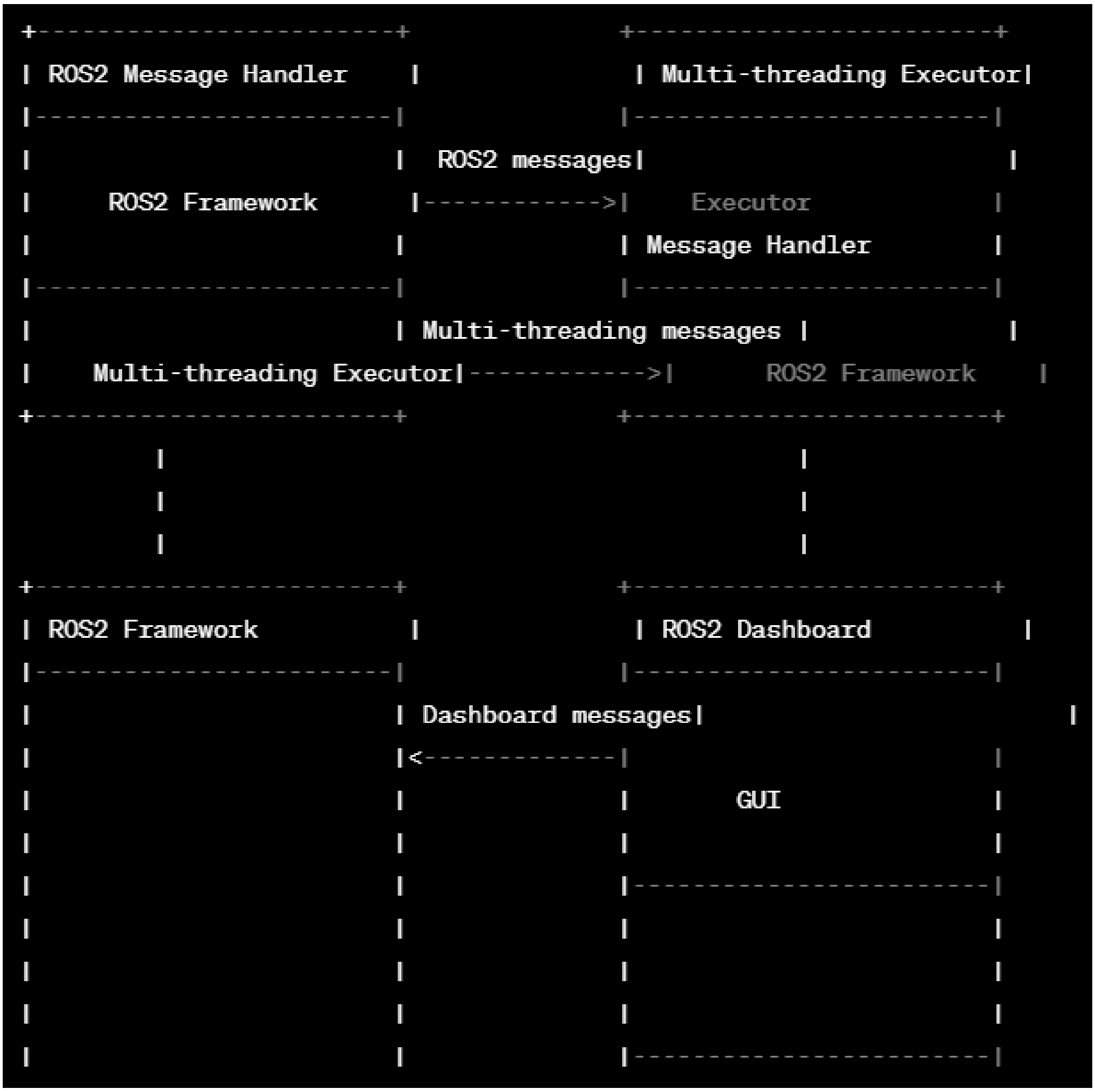
Summarized Data Types Transmitted between Components:



```
+-----------------------------------------------------+
|            ROS2 Framework & Executor                 |
+-----------------------------------------------------+
|                                                    |
|                 ROS2 messages                        |
|                                                    |
+-----------------------------------------------------+
                           |
                           |
+-----------------------------------------------------+
|        Multi-threading Executor & Executor           |
+-----------------------------------------------------+
|                                                    |
|             Multi-threading messages                 |
|                                                    |
+-----------------------------------------------------+
                           |
                           |
+-----------------------------------------------------+
|    ROS2 Framework & ROS2 Dashboard                   |
+-----------------------------------------------------+
|                                                    |
|               Dashboard messages                     |
|                                                    |
+-----------------------------------------------------+
```

Interfaces:

- ROS2 Message Handler provides an interface to the ROS2 Framework for handling ROS2 messages.
- Executor Message Handler provides an interface to the Multi-threading Executor for handling multi-threading messages.
- The ROS2 Framework provides an interface to the Multi-threading Executor for passing messages.
- The ROS2 Framework provides an interface to the ROS2 Dashboard for passing dashboard messages.
- The ROS2 Dashboard provides an interface to the ROS2 Framework for displaying data.



```
+------------------------+              +------------------------+
| ROS2 Message Handler    |              | Multi-threading Executor|
|------------------------|              |------------------------|
|                        |  ROS2 messages|                        |
|     ROS2 Framework      |------------>|    Executor            |
|                        |              | Message Handler        |
|------------------------|              |------------------------|
|                        | Multi-threading messages |             |
|    Multi-threading Executor|------------>|      ROS2 Framework    |
+------------------------+              +------------------------+
         |                                          |
         |                                          |
         |                                          |
+------------------------+              +------------------------+
| ROS2 Framework          |              | ROS2 Dashboard          |
|------------------------|              |------------------------|
|                        | Dashboard messages|                        |
|                        |<-------------|                        |
|                        |              |       GUI              |
|                        |              |                        |
|                        |              |------------------------|
|                        |              |                        |
|                        |              |                        |
|                        |              |                        |
|                        |              |------------------------|
```