

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.

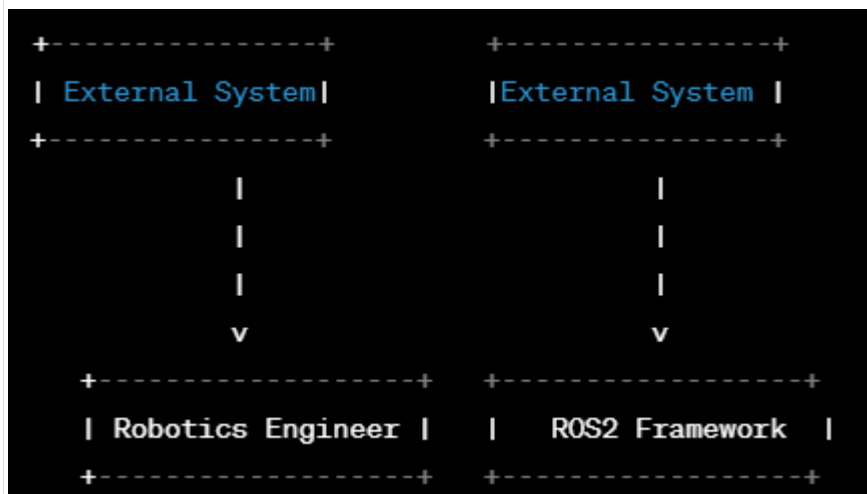
The problem is that robotics engineers lack an intuitive way to utilize multi-core processors to lower execution time in the Robot Operating System 2 (ROS) framework, which requires specialized knowledge of concurrency and low-level details of multi-threading. This leads to

decreased efficiency and productivity. The solution is to develop a software system architecture that provides a framework to facilitate the development of multi-threaded applications in ROS2, with functional requirements such as pre-locking heap memory, process priority configuration, measurement of heap allocations, latency, jitter, CPU caches usage, and memory fetches, executor type, node to process assignment, allocator type, easy-to-use user interface, guidelines for parallelizing the code, and parallel configuration. Quality requirements include increased CPU utilization, decreased heap allocation, increased general performance of the application, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity of robotics engineers, usability, and modularity. The use cases include direct inverse kinematics, detecting human faces and torso orientation, Kalman filter, and assessing execution time of real-time loop.

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 note that as a language model, I am not able to generate visual diagrams, but I can provide a textual representation of the architecture as follows:

System Context Diagram:



External Stakeholders:

- Robotics Engineer

External Systems:

- External System 1
- External System 2

Abstract Data Types:

- Data related to robotics applications
- Data related to multi-threading

Description of Components and Connections:

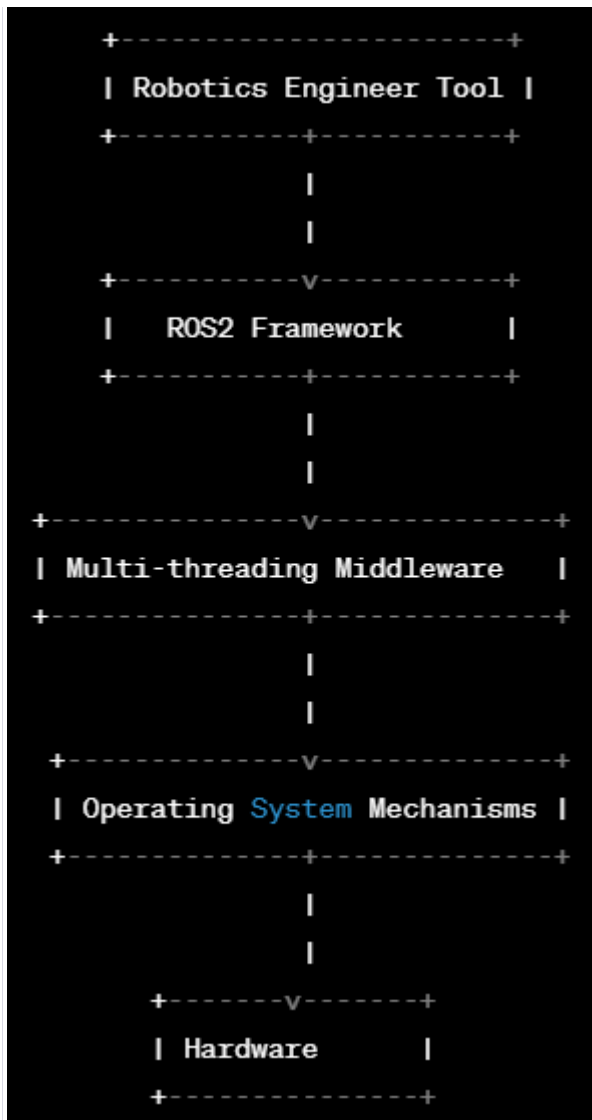
- **Robotics Engineer:** This is the external stakeholder who uses the ROS2 Framework to develop robotics applications. They provide input data related to the application and receive output data related to the performance of the application.
- **ROS2 Framework:** This is the software system architecture solution that facilitates the development of multi-threaded applications in ROS2. It receives input data related to the robotics application from the Robotics Engineer, processes the data using multi-threading, and provides output data related to the performance of the application to the Robotics Engineer. The framework also interacts with external systems as needed.
- **External System 1 and 2:** These are any external systems that the ROS2 Framework needs to interact with in order to provide its functionality.

The main data types transferred between the Robotics Engineer and the ROS2 Framework are related to the robotics application and multi-threading. The input data related to the application includes information about the nodes and their configuration. The output data related to the performance of the application includes measurements of heap allocations, latency, jitter, CPU caches usage, and memory fetches. The framework also provides output data related to the best performing executor type configuration containing nodes, node assignment, thread, process, and allocator type.

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 note that as a language model, I am not able to generate visual diagrams, but I can provide a textual representation of the architecture as follows:

Container Diagram:



How the main stakeholder will use the solution: The Robotics Engineer Tool component is used by the Robotics Engineer to develop robotics applications with the help of the ROS2 Framework. The Robotics Engineer Tool provides input data related to the application and receives output data related to the performance of the application.

Important Quality Properties and Adequate Architectural Solutions:

- Increased CPU Utilization: To address this quality property, the multi-threading middleware component will use a concurrent programming architectural style and design tactics such as thread pooling and load balancing.
- Decreased Heap Allocation: To address this quality property, the ROS2 Framework will use an object pooling design pattern to reduce heap allocation and reuse objects.
- Increased General Performance: To address this quality property, the ROS2 Framework will use a publish-subscribe architectural style to reduce coupling between nodes and increase modularity.
- Decreased Average Execution Time: To address this quality property, the multi-threading middleware component will use a concurrent programming architectural style and design tactics such as thread pooling and load balancing.

- Increased CPU Cache Utilization and Decreased Memory Fetches: To address these quality properties, the ROS2 Framework will use a data locality design pattern and optimize cache usage by grouping data that is frequently accessed together.
- Increased Productivity of Robotics Engineers: To address this quality property, the Robotics Engineer Tool will have an easy-to-use user interface and provide guidelines for parallelizing code.

Data Types Transferred Between Components:

- Data related to robotics applications
- Data related to multi-threading
- Memory allocation data
- Latency data
- Jitter data
- CPU cache usage data
- Memory fetches data
- Node configuration data

Interfaces Between Components:

- Robotics Engineer Tool: Provides input data related to the application, such as node configuration data. Requires output data related to the performance of the application, such as memory allocation data and latency data.
- ROS2 Framework: Receives input data related to the application from the Robotics Engineer Tool, such as node configuration data. Provides output data related to the performance of the application to the Robotics Engineer Tool, such as memory allocation data and latency data. Requires data related to multi-threading from the Multi-threading Middleware component.
- Multi-threading Middleware: Provides data related to multi-threading to the ROS2 Framework. Requires data related to operating system mechanisms from the Operating System Mechanisms component.
- Operating System Mechanisms: Provides data related to operating system mechanisms to the Multi-threading Middleware component.
- Hardware: Provides the underlying hardware resources needed to run the system.

Data Types Transmitted Between Components:

- Input data related to the application, such as node configuration data
- Output data related to the performance of the application, such as memory allocation data and latency data
- Data related to multi-threading
- Data related to operating system mechanisms

Anti-patterns to Avoid:

- God Object anti-pattern: Avoid creating a single component that handles all functionality. Instead, use a modular design with well-defined interfaces between components.
- Spaghetti Code anti-pattern: Avoid creating code that is difficult to understand and maintain. Instead, use design patterns

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 note that as a language model, I am not able to generate visual diagrams, but I can provide a textual representation of the architecture as follows:

Component Diagram:

```
+-----+
| Robotics Engineer |
+-----+-----+
|
|
+-----v-----+
| Robotics Engineer Tool |
+-----+-----+
|
|
+-----v-----+
| ROS2 Framework |
+-----+-----+
|
|
+-----v-----+
| Performance |
| Evaluation Tool |
+-----+-----+
|
|
+-----v-----+
| Multi-threading |
| Middleware |
+-----+-----+
|
|
+-----v-----+
| Operating System |
| Mechanisms and Tools |
+-----+-----+
|
|
+-----v-----+
| Hardware |
+-----+
```


How the main stakeholder will use the solution: The Robotics Engineer will use the Robotics Engineer Tool to develop robotics applications. The Performance Evaluation Tool will be used to measure the performance of the application.

Important Quality Properties and Adequate Architectural Solutions:

- Increased CPU Utilization: To address this quality property, the Multi-threading Middleware component will use a concurrent programming architectural style and design tactics such as thread pooling and load balancing.
- Decreased Heap Allocation: To address this quality property, the ROS2 Framework will use an object pooling design pattern to reduce heap allocation and reuse objects.
- Increased General Performance: To address this quality property, the ROS2 Framework will use a publish-subscribe architectural style to reduce coupling between nodes and increase modularity.
- Decreased Average Execution Time: To address this quality property, the Multi-threading Middleware component will use a concurrent programming architectural style and design tactics such as thread pooling and load balancing.
- Increased CPU Cache Utilization and Decreased Memory Fetches: To address these quality properties, the ROS2 Framework will use a data locality design pattern and optimize cache usage by grouping data that is frequently accessed together.
- Increased Productivity of Robotics Engineers: To address this quality property, the Robotics Engineer Tool will have an easy-to-use user interface and provide guidelines for parallelizing code.

Data Types Transferred Between Components:

- Data related to robotics applications
- Data related to multi-threading
- Memory allocation data
- Latency data
- Jitter data
- CPU cache usage data
- Memory fetches data
- Node configuration data

Interfaces Between Components:

- Robotics Engineer: Provides input data related to the application, such as node configuration data. Receives output data related to the performance of the application, such as memory allocation data and latency data.
- Robotics Engineer Tool: Receives input data related to the application from the Robotics Engineer, such as node configuration data. Provides output data related to the performance of the application to the Performance Evaluation Tool, such as memory allocation data and latency data. Requires data related to multi-threading from the Multi-threading Middleware component.
- ROS2 Framework: Receives input data related to the application from the Robotics Engineer Tool, such as node configuration data. Provides output data related to the performance of the application to the Performance Evaluation Tool, such as memory

allocation data and latency data. Requires data related to multi-threading from the Multi-threading Middleware component.

- Performance Evaluation Tool: Receives input data related to the performance of the application from the Robotics Engineer Tool and ROS2 Framework, such as memory allocation data and latency data. Provides output data related to the performance of the application to the Robotics Engineer Tool, such as recommendations for improving performance.
- Multi-threading Middleware: Provides data related to multi-threading to the ROS2 Framework. Requires data related to operating

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.

To implement the proposed solution, the following toolchain can be used:

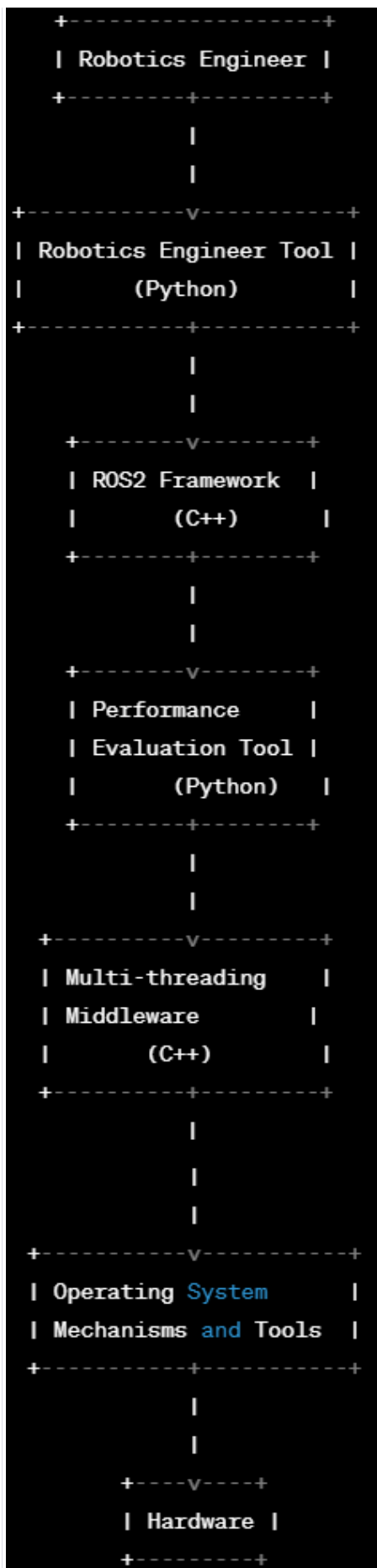
Programming Languages and Frameworks:

- ROS2 Framework: C++
- Multi-threading Middleware: C++
- Robotics Engineer Tool: Python
- Performance Evaluation Tool: Python

Tools to Reuse:

- Operating System Mechanisms and Tools: Use existing operating system mechanisms and tools to implement the Multi-threading Middleware component.

Toolchain Diagram:



Data Types Transferred Between Components:

- ROS2 messages
- Performance data

Interfaces Between Components:

- Robotics Engineer: Provides input data related to the application, such as ROS2 messages. Receives output data related to the performance of the application, such as performance data.
- Robotics Engineer Tool: Receives input data related to the application from the Robotics Engineer, such as ROS2 messages. Provides output data related to the performance of the application to the Performance Evaluation Tool, such as performance data. Requires data related to multi-threading from the Multi-threading Middleware component.
- ROS2 Framework: Receives input data related to the application from the Robotics Engineer Tool, such as ROS2 messages. Provides output data related to the performance of the application to the Performance Evaluation Tool, such as performance data. Requires data related to multi-threading from the Multi-threading Middleware component.
- Performance Evaluation Tool: Receives input data related to the performance of the application from the Robotics Engineer Tool and ROS2 Framework, such as performance data. Provides output data related to the performance of the application to the Robotics Engineer Tool, such as recommendations for improving performance.
- Multi-threading Middleware: Provides data related to multi-threading to the ROS2 Framework. Requires data related to operating system mechanisms from the Operating System Mechanisms and Tools component.