

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

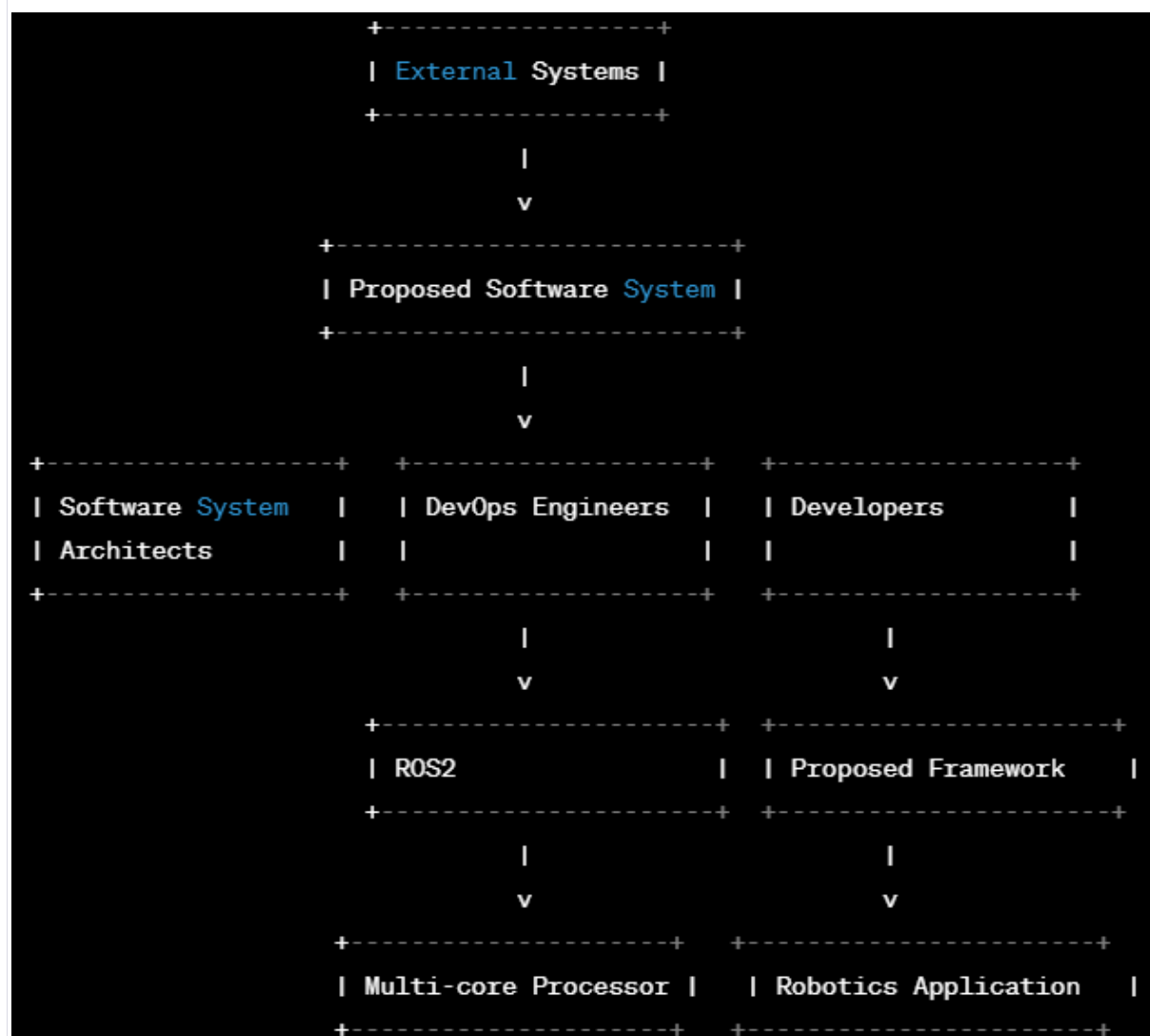
the robotics engineers • SR.05: Measurement of latency. - The framework has a mechanism that measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: 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 of utilizing multi-cores to reduce execution time, and they need to know low-level details of parallelization and develop multi-threading applications. The solution is to develop a software framework that assists robotics

engineers in developing software that takes advantage of multi-cores, integrated into ROS2. The framework should provide functionality for pre-locking heap memory, process priority configuration, measurement of heap allocations, latency, jitter, CPU caches usage, and memory fetches. It should also provide guidance on parallelizing code and advise on the best performing executor type, node to process assignment, and allocator type. The 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, 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.

Here is a C4 system context diagram for the proposed software system architecture solution:



The diagram shows the external systems as well as the external stakeholders involved in the proposed software system architecture solution. The proposed software system consists of two main components, namely ROS2 and the proposed framework.

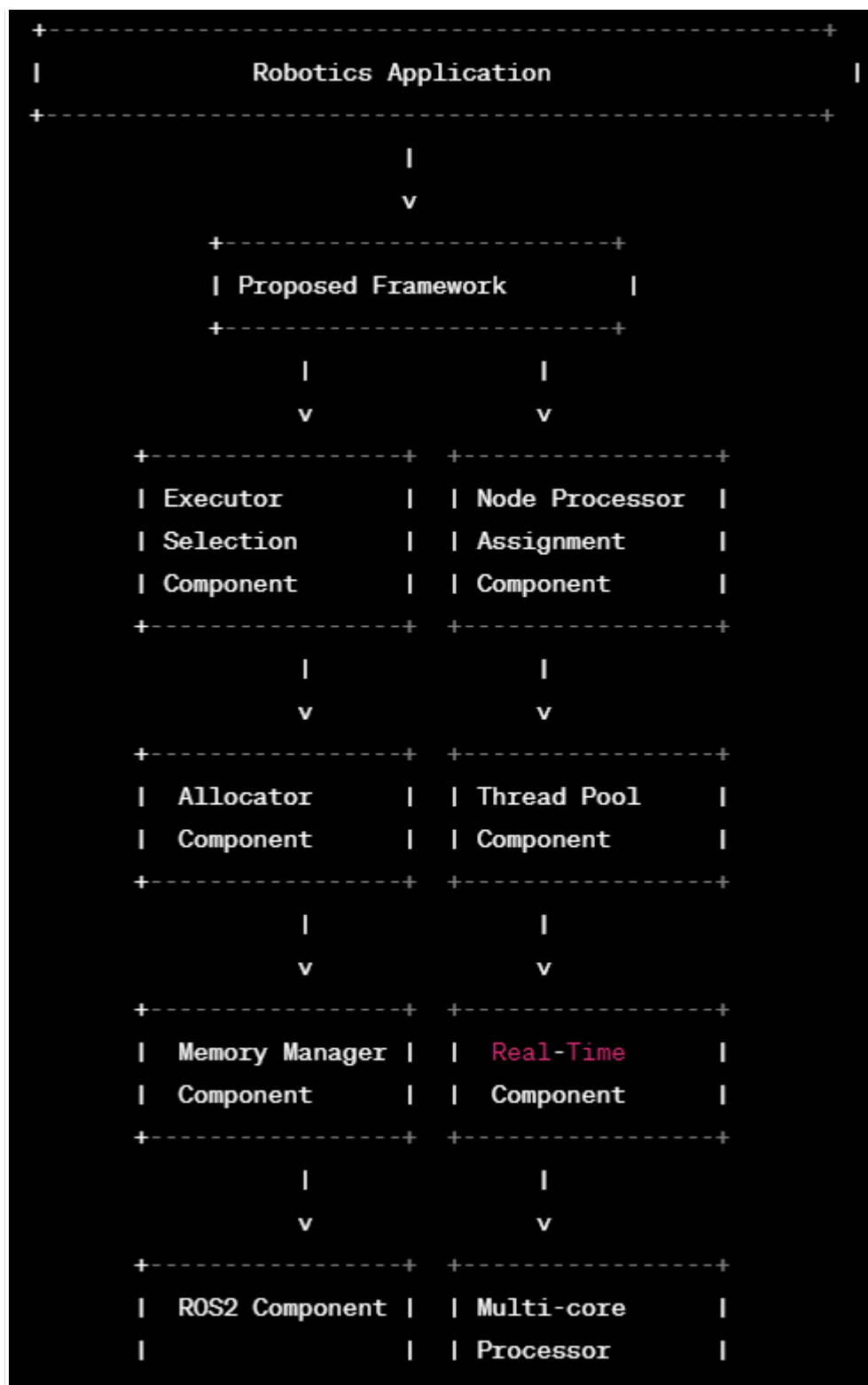
The external stakeholders include software system architects, DevOps engineers, and developers. They interact with the software system by utilizing the proposed framework to develop robotics applications that take advantage of multi-core processors. The software system architects are responsible for designing the overall architecture of the software system, while DevOps engineers are responsible for ensuring that the software system runs smoothly in a production environment. Developers are responsible for writing the code that runs on the software system.

The proposed framework receives data types such as nodes' configuration as input from the developers and outputs the best performing executor type configuration containing nodes, node assignment, thread, process, and parallel configuration guidelines to the developers. The ROS2 component interacts with the multi-core processor and the robotics application, receiving and processing data from them.

Overall, the proposed software system architecture solution aims to assist robotics engineers in developing software that takes advantage of multi-core processors, integrated into ROS2.

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 proposed software system architecture solution:



The main stakeholders, software system architects, DevOps engineers, and developers, will use the proposed software system architecture solution by utilizing the proposed framework to develop robotics applications that take advantage of multi-core processors. The software system architects are responsible for designing the overall architecture of the software system, while DevOps engineers are responsible for ensuring that the software system runs smoothly in a production environment. Developers are responsible for writing the code that runs on the software system.

The quality properties that are important include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity, usability, and modularity. To address these quality properties, the proposed solution incorporates the following architectural components:

- **Executor Selection Component:** This component selects the best performing executor type configuration containing nodes, node assignment, thread, process, and parallel configuration guidelines. It uses design patterns like the Strategy pattern to allow for easy selection of the best-performing executor type.
- **Node Processor Assignment Component:** This component assigns nodes to threads and processes to achieve the best performance. It uses the design patterns like Chain of Responsibility pattern to enable easy assignment of nodes.
- **Allocator Component:** This component advises developers on which allocator to use for message data types used throughout the application. It uses design patterns like the Template pattern to allow for easy advice on which allocator to use.
- **Thread Pool Component:** This component provides a pool of threads to be used by the nodes in the application. It uses the design patterns like the Object Pool pattern to allow for easy management of threads.
- **Memory Manager Component:** This component manages memory in the application by pre-locking heap memory during initialization to avoid failed memory allocations. It uses the design patterns like the Singleton pattern to ensure that the memory is managed in one location.
- **Real-Time Component:** This component measures and improves the performance of the real-time loop. It uses design patterns like the Observer pattern to notify the developers of the real-time performance of the application.
- **ROS2 Component:** This component interacts with the multi-core processor and the robotics application, receiving and processing data from them. It uses the architectural style of client-server to interact with the robotics application and the multi-core processor.

The data types transmitted between components include nodes' configuration, executor type configuration, allocator type, and message data types. The interfaces between components are defined as follows:

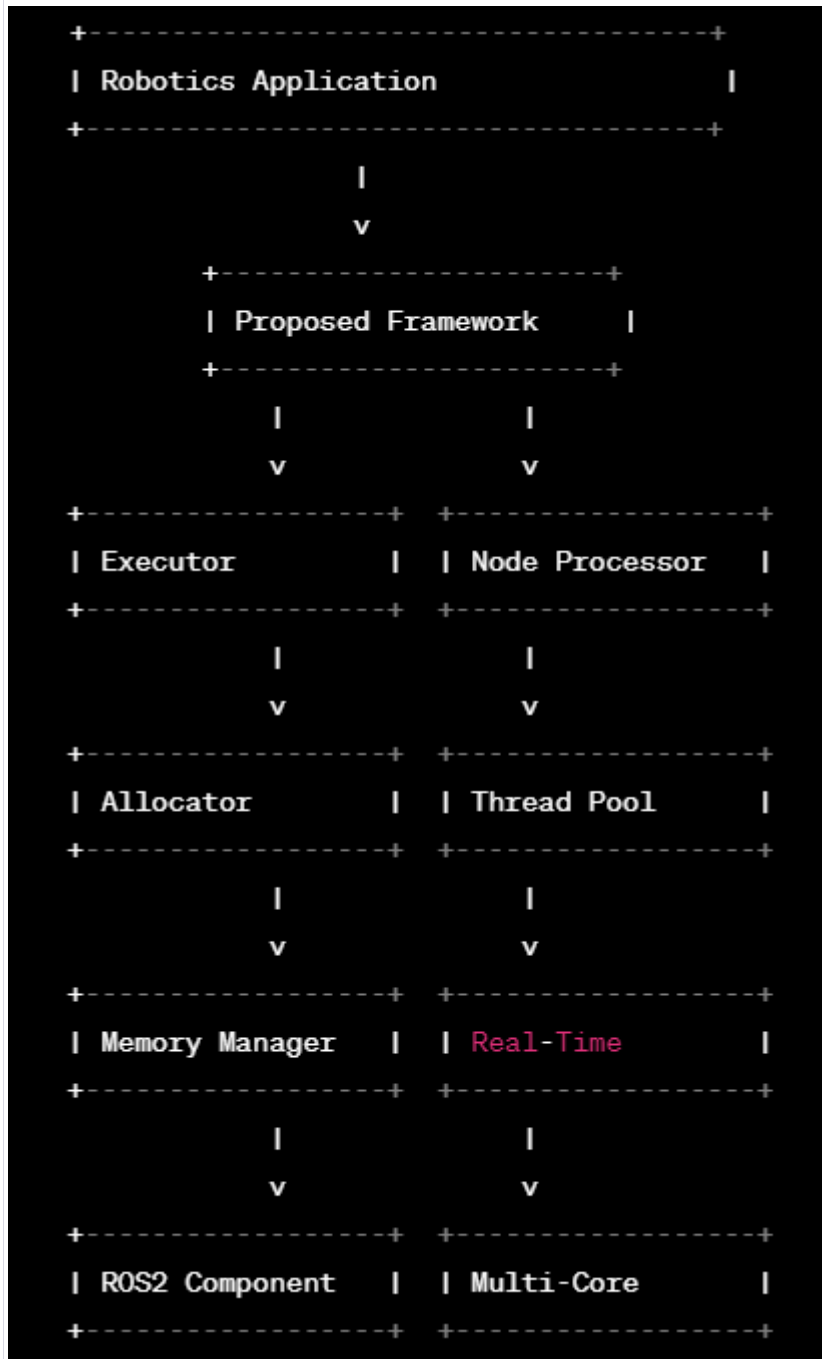
- Executor Selection Component provides the Executor Selection interface to the Node Processor

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 proposed software system architecture solution:



The main stakeholders, software system architects, DevOps engineers, and developers, will use the proposed software system architecture solution by utilizing the proposed framework to develop robotics applications that take advantage of multi-core processors.

The quality properties that are important include increased CPU utilization, decreased heap allocation, increased general performance, decreased average execution time, increased CPU cache utilization, decreased memory fetches, increased productivity, usability, and modularity. To

address these quality properties, the proposed solution incorporates the following architectural components:

- **Executor Component:** This component implements the Executor Selection interface and selects the best-performing executor type configuration containing nodes, node assignment, thread, process, and parallel configuration guidelines. It uses design patterns like the Strategy pattern to allow for easy selection of the best-performing executor type.
- **Node Processor Component:** This component implements the Node Processor Assignment interface and assigns nodes to threads and processes to achieve the best performance. It uses the design patterns like Chain of Responsibility pattern to enable easy assignment of nodes.
- **Allocator Component:** This component implements the Allocator interface and advises developers on which allocator to use for message data types used throughout the application. It uses design patterns like the Template pattern to allow for easy advice on which allocator to use.
- **Thread Pool Component:** This component provides a pool of threads to be used by the nodes in the application. It uses the design patterns like the Object Pool pattern to allow for easy management of threads.
- **Memory Manager Component:** This component implements the Memory Manager interface and manages memory in the application by pre-locking heap memory during initialization to avoid failed memory allocations. It uses the design patterns like the Singleton pattern to ensure that the memory is managed in one location.
- **Real-Time Component:** This component implements the Real-Time interface and measures and improves the performance of the real-time loop. It uses design patterns like the Observer pattern to notify the developers of the real-time performance of the application.
- **ROS2 Component:** This component implements the ROS2 interface and interacts with the multi-core processor and the robotics application, receiving and processing data from them. It uses the architectural style of client-server to interact with the robotics application and the multi-core processor.

The data types transmitted between components include nodes' configuration, executor type configuration, allocator type, and message data types. The interfaces between components are defined as follows:

- Executor Component provides the Executor Selection interface to the Node Processor Component.
- Node Processor Component provides the Node Processor Assignment interface to the Executor Component and the Thread Pool Component.
- Allocator Component provides the Allocator interface to the ROS2 Component.
- Thread Pool Component provides the Thread Pool interface to the Node Processor Component.
- Memory Manager Component provides the Memory Manager interface to the ROS2 Component.
- Real-Time Component provides the Real-Time interface to the ROS2 Component

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.

Here is a suggested toolchain for implementing the proposed software system architecture solution:

- ROS2: This will be the foundation of the solution, as it is the robotics engineering framework that the proposed solution is built upon.
- C++: This will be the primary programming language for developing the proposed framework and its components. C++ is a commonly used language for robotics engineering, and it is well-suited for low-level development tasks like parallelization and memory management.
- Boost: This C++ library provides many useful tools for parallel programming, including multi-threading and concurrency utilities. It can be used to simplify the development of the proposed framework.
- OpenCV: This computer vision library can be used to implement the image processing algorithms required for the "Detecting human faces and torso orientation" use case.

The data types transferred between components include nodes' configuration, executor type configuration, allocator type, and message data types. The interfaces between components and the data types they require and provide are defined as follows:

- Executor Component provides the Executor Selection interface, which requires nodes' configuration and provides executor type configuration.
- Node Processor Component provides the Node Processor Assignment interface, which requires nodes' configuration and provides thread and process assignments.
- Allocator Component provides the Allocator interface, which requires message data types and provides allocator types.
- Thread Pool Component provides the Thread Pool interface, which requires thread assignments.
- Memory Manager Component provides the Memory Manager interface, which requires memory allocation information.
- Real-Time Component provides the Real-Time interface, which requires real-time loop execution time, latency, jitter, CPU cache usage, and memory fetches information.