CISC 322

Assignment 3: Report

Apollo Proposed Feature Enhancement

Team Artemis

Josh Otten (18jbo@queensu.ca)

Aleks Jugovic (16anj@queensu.ca)

Muyun Yang (18my24@queensu.ca)

Chong Guan (18cg14@queensu.ca)

Daniel Jang (17ddhj@queensu.ca)

Wooseok Lee (20wl9@queensu.ca)

Abstract

This report is to propose a feasible enhancement on the current concrete architecture of Apollo, and further explore why we are concerned about that and what effect would this improvement bring to the system and the architectures once implemented.

In the first two reports, we went through the process of discovering conceptual and concrete architectures. The salient finding in the first report was identifying the Publish-Subscribe style of architecture. Benefitting from the style, all the subsystems are able to cooperate with each other in terms of publishing and listening to messages from subscribed modules. While in the second report, we pursued a more detailed and extensive investigation of the Apollo system, which offered us a revealing insight into the architecture and the functioning of modules.

In the scope of this report, considering the importance of safe driving and the non-practical implementation of the Guardian module, we proposed an enhancement to it. The enhancement is based on the Pub-Sub style of architecture. The Guardian module becomes the monitor of all other modules. It subscribes to the failure message reported by the Monitor module and takes reasonable actions on the basis of the physical environment. To accomplish the enhancement, we suggested two approaches. A Software Architecture Analysis Method was carried out to determine which one is better regarding non-functional requirements and stakeholders.

Either approach has no impact on the overall style of conceptual architecture but both influence the functionality of the system and dependencies in the concrete architecture to some extent.

Table of Contents

Introduction

This report is aimed at examining the current Apollo system, figuring out shortcomings in the system and introducing a constructive modification to it. When we reviewed the Apollo source code [1], we realized that the Guardian module is simply implemented as a message router, which detects failure messages and directly sends an immediate-stop message to control modules. It could result in implicit and severe threats to drivers' safety but gives us a window to optimize the Guardian module.

To strengthen the ability to handle unpredictable situations, the Guardian module is supposed to have more interactions with sensing, planning, and controlling modules. In both approaches considered, Guardian caches data from other subsystems to use in creating a new path to safely stop the vehicle. In the first approach, different types of handlers in the Guardian module are designed to process error messages before stopping the car. Failure messages of different modules will initialize a series of handlers to deal with the corresponding scenarios. The second approach gives us a more moderate way of coping with emergencies. The Guardian module assesses a level of risk to the failed module, and brings the car to a stop accordingly. It does this by either using a scenario from Storytelling, creating a new stopping path with Planning, or immediately stopping the car.

By doing the SAAM analysis, the second approach won our trust, which is mostly owing to the high performance and efficiency when adopting the second approach. Regarding the SAAM analysis, we examined approaches from two aspects, namely NFRs (categorized into Performance, Safety, Maintainability, Robustness, Testability) and Stakeholders (categorized into Driver, Safety Evaluators, Module Programmers, Insurers).

After performing the analysis on the second approach, we found that it makes changes to the concrete architecture. Because the Guardian module subscribes to several modules, it introduces new dependencies and subcomponents to itself. At the end of the report, we provide three test plans on different scenarios.

Proposed Enhancement

The proposed enhancement is for the Guardian subsystem of Apollo. The Guardian module [1] brings the vehicle to a stop if the Monitor module detects a module failure, either applying a hard brake, a soft brake, or in special cases, waiting 10 seconds for the driver of the vehicle to intervene before bringing the vehicle to a stop. The proposed enhancement is to enhance these stopping scenarios. Depending on which module failure is reported by Monitor, Guardian will attempt to bring the car to a safe stop. Instead of blindly applying the brakes, Guardian will instead create a new trajectory to bring the vehicle out of the way of any potential danger before bringing the car to a stop.

Guardian will accomplish this by caching previous information from the vehicle's modules. This is done to provide enough information about the environment of the vehicle moments before the detected module failure. The cached information will be used in place of real-time information to create a new vehicle trajectory. For example, if the vehicle is in traffic, Guardian will attempt to pull the vehicle over to the side of the road before coming to a stop. In

theory, the cached information from moments before the detected failure is enough to accomplish this.

Approach A

The first approach for the enhancement can be seen in Figure 1. This approach would make a few changes to the high-level concrete architecture of Apollo. First, Guardian would subscribe to all other subsystems in Apollo in order to receive information to make an informed decision. It also would publish data to the HMI in order to inform the driver about the decision it made when making a stop.

The internal conceptual architecture of Guardian would have many new components and dependencies with the first approach of the enhancement. Overall, this approach would use an object-oriented architectural style for its benefits with modifiability and extensibility. The Main Component would receive messages from Monitor and once it receives a failure message, it would instantiate the Failure Handler with the failure message received. The Failure Handler would determine which subsystem has failed and instantiate the Scenario Handler with this information. The Scenario handler would be implemented as a hierarchy and have an implementation for every different subsystem, with the correct concrete class invoked based on the failure reason. The Scenario Handler would read from the Data Cache to determine the best form of a stop based on internal algorithms, then instantiate the Controller with a series of commands to send to the CanBus to stop the vehicle.

Overall, this approach has the benefit of being extendable and robust due to the object-oriented design and a large amount of data it factors into stopping decisions. However, it would be difficult to maintain and test due to the amount of new dependencies being added, and thus we ultimately decided to go with approach B for the feature enhancement.

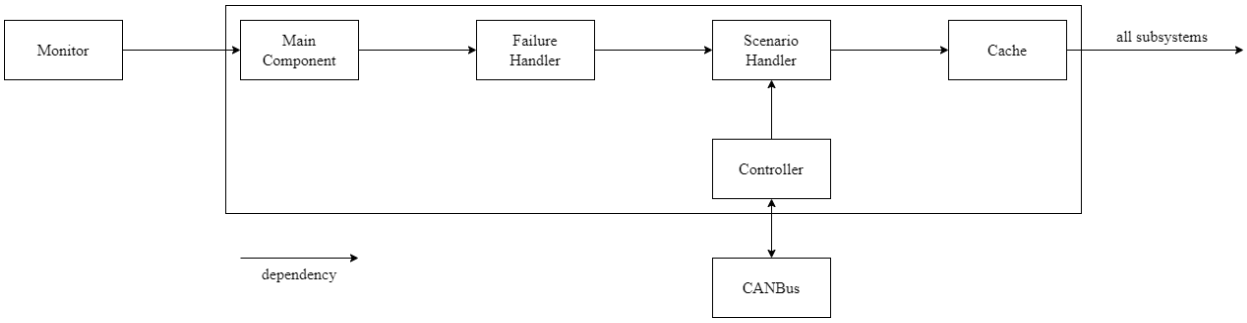

Figure 1. Approach A Implementation of the new Guardian subsystem and impacted subsystem

Approach B

The second approach for the enhancement can be seen in Figure 2. This approach makes fewer changes to the overall high-level concrete architecture of Apollo. The Guardian subsystem subscribes to Monitor, Control, CanBus, Storytelling, Localization, Perception, and Prediction. Additionally, the Storytelling and Planning subsystems subscribe to Guardian. Guardian will receive information from the subsystems it is subscribed to, and publish messages which tell the new connected subsystems to create a new route. This approach assumes that only a single subsystem fails and that other subsystems are working properly.

The internal conceptual architecture of Guardian would have new components that expand upon its basic functionalities. These new components would follow a Publish-Subscribe architectural style. The Cache component caches data from Routing, Localization, Perception, and Prediction, and stores all the data from these components for the previous 5-10 seconds. The Proc component constantly polls the Module subsystem to check for any module failures. If no failure is detected, the PassThrough component passes through Control signals to CanBus. This functionality is exactly like the current functionality of Guardian.

When a failure is detected, the SafetyMode component activates. Depending on the risk level of the component failure, the vehicle is stopped in a different way. The risk level depends on whether or not the component is vital to the functioning of autonomous driving. Module failures that are low risk include Task Manager, Dreamview, HD Map, and Routing. When one of these modules fails, SafetyMode can instruct Planning to create a new trajectory, or use a scenario from Storytelling to bring the vehicle to a stop at the side of the road. Medium risk failures are that of Storytelling, Localization, Perception, and Prediction. Should these modules fail, Apollo cannot create a trajectory based on real-time information. Instead, SafetyMode will send information from Cache to Planning in order to create a new trajectory based on information from just before a module failure is detected. High-risk failures are that of Planning, CanBus, Monitor, and Control. In this case, a new trajectory cannot be created, or there is a hardware error. The vehicle will be brought to an immediate stop. This is the functionality already present in Guardian.

Overall, this approach has a smaller impact on the overall architecture of Apollo than Approach A, while still retaining the functionalities already existing in the unmodified Guardian subsystem. Instead of having many failure scenarios for each module, modules are grouped together in an attempt to lower the number of possible cases to consider.

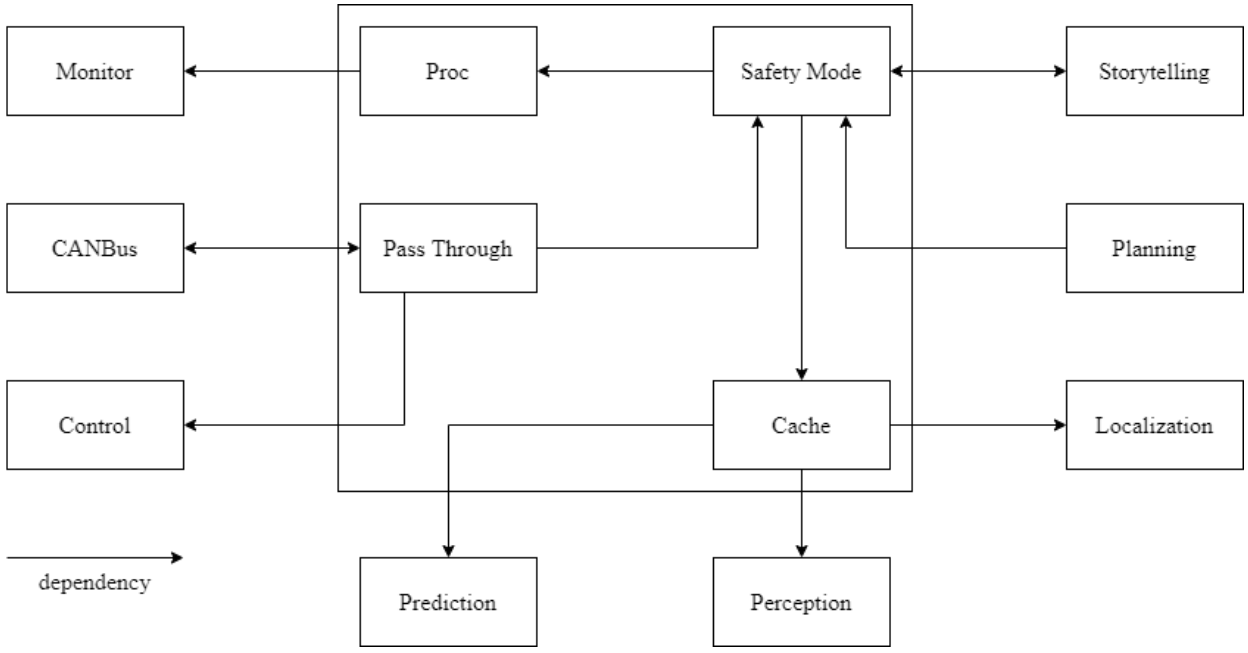

Figure 2. Approach B Implementation of the new Guardian subsystem and impacted subsystems

Motivation

As it stands, the Guardian subsystem simply polls the Monitor subsystem for any module failures. If there is no failure detected, Guardian simply passes through Control signals to CanBus. If a module failure is detected, Guardian will bring the vehicle to a stop. There are three scenarios to stop the vehicle, but they are not very sophisticated. In each one, Guardian simply applies the brakes and brings the car to a stop without much consideration for the environment of the vehicle.

Guardian is a module that only has one, relatively straightforward source code file. In discussing potential areas of improvement for Apollo, this stood out. In Apollo's documentation, Guardian is described briefly as a safety module, but lacks substance. There are many scenarios that must be considered when developing an autonomous vehicle system. What if there is a module failure while the car is in traffic? Guardian will apply the brakes, bringing the vehicle to a stop. Disrupting the flow of traffic by sudden braking can be incredibly dangerous. Instead, pulling over at the side of the road is a much safer stop.

SAAM analysis

Stakeholder	Most important NFRs regarding the enhancement
Driver	*Performance:* The new enhancement should bring better performance in most cases. *Usability:* The car should inform the user of what action it's performing. *Safety:* Movements should be as smooth as possible.
Safety Evaluators	*Safety:* The car needs to choose the safest way to stop.
Module Programmers	*Testability:* The module should be programmed in such a way that it can be tested in a safe environment and/or virtually. The programmers are also looking for a testing environment that is easy to maintain and modify. *Maintainability:* The module should be easy to maintain, especially when new modules are added that the guardian needs to check.
Insurers	*Robustness:* This approach must work properly under adverse conditions. *Safety:* The car should avoid accidents as much as possible.

Table 1. Impacts of approach on Stakeholders

Attribute	Approach A	Approach B
Performance: This approach should have a low response time when dealing with many inputs from its environment and handling failures in different challenging weather and traffic conditions.	*Low*. This approach chooses from many plans depending on which module fails and is slower compared to the other approach, due to the bottleneck in the failure handler.	*High*. This approach requires only selecting from three different plans, which is much quicker and effective enough in all different situations.
Safety: This approach needs to identify errors, avoid accidents, notify users in dangerous situations, and allow users to take control when the system malfunctions.	*Moderate*. This approach uses information on which module failed to choose the safest way to stop, allowing many options for each error. However, because of its huge depository, it may be too slow when the accident happens.	*Moderate*. This approach uses the risk level to choose the safest way to stop, which can reduce the risk drastically: the higher the risk level, the less chance of making an unsafe decision. However, there is a lack of preparation for some extreme situations.
Maintainability: This enhancement should be easily maintained by developers.	*Low*. This approach has to deal with every subsystem, is tedious to maintain, and has a high time cost.	*High*. This approach only has to deal with three risk levels, which is much easier for the developers.
Robustness: This approach must work properly under adverse conditions.	*High*. This approach has an individual plan for each module failure scenario, which allows it to be really flexible and could be working in all different situations, making it robust.	*Moderate*. This approach uses three different plans for all situations and works in most cases. However, due to the lack of options, sometimes a plan for each risk level may not be the best method in this scenario.
Testability: In this approach, every scenario should be easily testable.	*Low*. This approach needs to be tested for each scenario and subsystem. In other words, a huge amount of resources is needed to cover every environment, making it hard to test.	*High*. This approach has only three main scenarios (low, medium, and high risk), which groups up subsystems that are impacted by each risk level. This allows an efficient and easy environment for testing as the enhancement of Guardian and the failure of other subsystems are grouped up in different risk levels.

Table 2. Impacts of each approach on NFRs

We decided to go with approach B. One reason is that it would have better performance and be more efficient since there are only three scenarios to choose from, so it can come to a decision quicker which could save precious time. It would also have better maintainability and testability: The approach only has to deal with three risk levels, so there are only three scenarios to maintain and test. When new scenarios are created, the developers could choose which risk level it belongs to and they could easily add the new scenario to that risk level which allows them to have great maintainability. Both approaches are robust as they can deal with most subsystem failures but approach A has higher robustness as it can have a distinct plan for each module failure which gives the system more flexibility on what each scenario can do. Although approach B has moderate robustness compared to approach A, we have decided to go with approach B as it is still robust enough and other significant non-functional requirements that were mentioned favours approach B.

Current state

The current conceptual architecture for Apollo can be seen in Figure 3. The main architectural style Apollo uses is publish-subscribe, where individual modules publish their messages and actions for other subsystems to subscribe to and receive in order to perform actions. Relative to the new enhancement proposed to Guardian, the arrows in red show the changes that were required to the conceptual architecture [2] in order for the enhancement to function. There are new dependencies involved with Guardian, Storytelling and Planning now subscribe to it, and Guardian will subscribe to Localization, Perception, and Prediction.

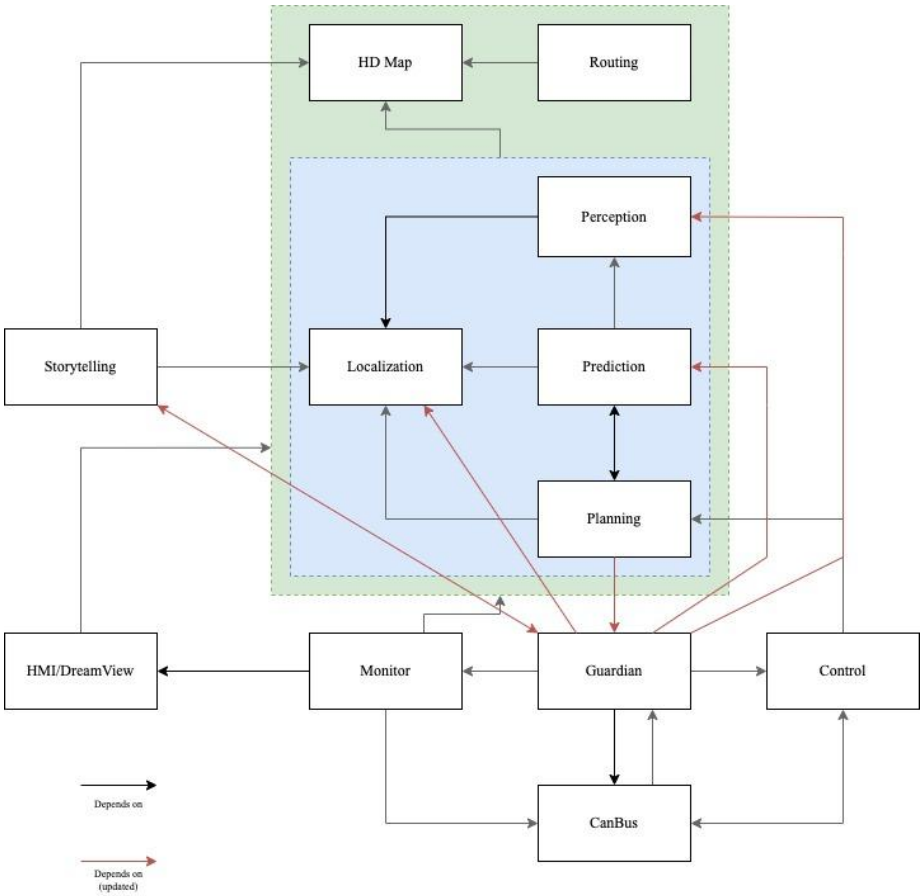

Figure 3. The current conceptual architecture with the new enhancement

The effects of the enhancement

The High- and Low-level Conceptual Architecture

On the high-level architecture, new connections are added between Guardian and the other subsystems to reflect the changes outlined in approach B. For low-level changes to Guardian's conceptual architecture, new subcomponents are introduced: Proc, Cache, SafetyMode, and PassThrough. Storytelling now includes new scenarios for Guardian. Planning can now accept cached data from Guardian in place of its usual inputs to create a new trajectory.

Performance

The overall performance of the system will decrease slightly due to the new overhead of constantly caching data from other subsystems.

Evolvability

There are three resources of evolution [3] which are changes in a system's environment (domain), requirements (experience) and implementation technologies (process). One example source that can cause the evolution of the new enhancement could be the organization that owns or utilizes Apollo or government regulations that causes an impact or changes to the scenario-handling responsibility of the Guardian subsystem. Another example would be the drivers who gain experience from using the autonomous vehicle and may have suggestions for the new enhancement, which in turn may cause the system to evolve. The new enhancement of the Guardian module is very flexible enough to handle new changes which can be added, modified, or removed for not only scenarios but the overall manipulation of the risk levels as well.

Testability & Maintainability

The new enhancement has a great effect on testability as the scenario to stop the car is now organized into three groups which makes it easier to test the Guardian subsystem.

Maintainability/Modifiability is positively impacted as well because the original Guardian subsystem only had 3 main scenarios that would stop the car and adding more scenarios would not be an organized task. As we happen to group them into different levels of risk, it is easier to add, edit, or remove scenarios easily. Modifying the subsystem also pairs nicely with updating the tests as well as you only need to focus on the changes of each level of risk.

Interactions with other features/subsystems

Guardian subscribes to Storytelling, Localization, Perception, and Prediction. From Storytelling, Guardian will read which stopping scenario will be used for low-risk failures. From Localization, Perception, and Prediction, Guardian will cache the information published by those subsystems in the Cache component. Storytelling and Planning subscribe to Guardian. PassThrough interacts with Control and CanBus normally, relaying Control instructions to CanBus. SafetyMode handles the failure conditions, either sending commands for new stopping

trajectories to Planning and Storytelling, or interrupting PassThrough and sending instructions to CanBus to bring the car to an immediate stop. Planning additionally will accept the data cached by Guardian from Cache to build this new trajectory.

Impacted directories and files

apollo/modules/guardian [1]: Add new subcomponents. Modify guardian_component.cc and guardian_component.h to accommodate new features.

apollo/modules/storytelling [1]: Modify to subscribe to messages from Guardian, and to include new Guardian-related stopping scenarios.

apollo/modules/planning [1]: Modify to subscribe to messages from Guardian to create a new trajectory, both with and without cached data.

Concurrency

A number of new concurrencies are introduced in the enhancement of Guardian. The Cache subsystem caches the output messages of Localization, Perception, and Prediction. PassThrough is constantly relaying Control signals to CanBus. Proc constantly checks for module failure messages from Monitor. All these processes are occurring concurrently in addition to the concurrencies that are already present within the Apollo architecture.

Use cases

Low-Risk Use Case

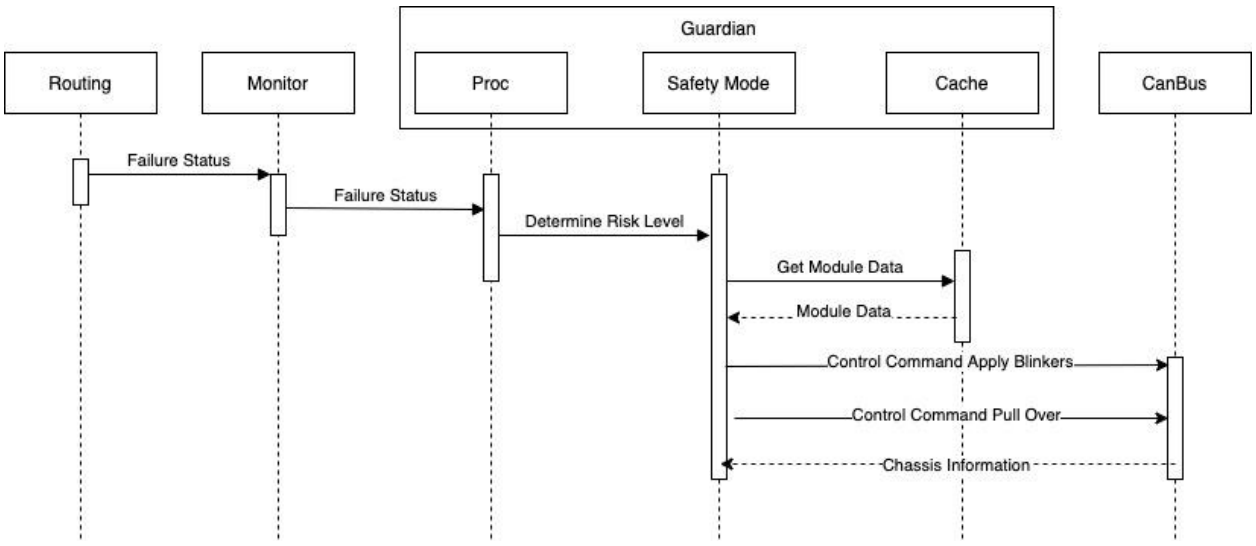

Figure 4. Sequence diagram of Low-Risk use case

In this use case, a low-risk module has failed and the Guardian will be able to perform a more sophisticated stop of the vehicle. The Routing module was used as the subsystem that failed as it is one that is deemed to be low risk in a failure situation. Routing will indicate that it has failed to the Monitor which will then propagate this message to Guardian. In Guardian, the

Proc subsystem will activate the safety mode and determine the level of risk by calling on the Safety Mode subsystem. Once it has been determined to be low risk, the Safety Mode module will grab cached data from the Cache to get all the required information it needs to make a stop. Then it will send a series of control commands to the CanBus to activate the blinkers of the car and pull over to the side of the road.

Medium-Risk Use Case

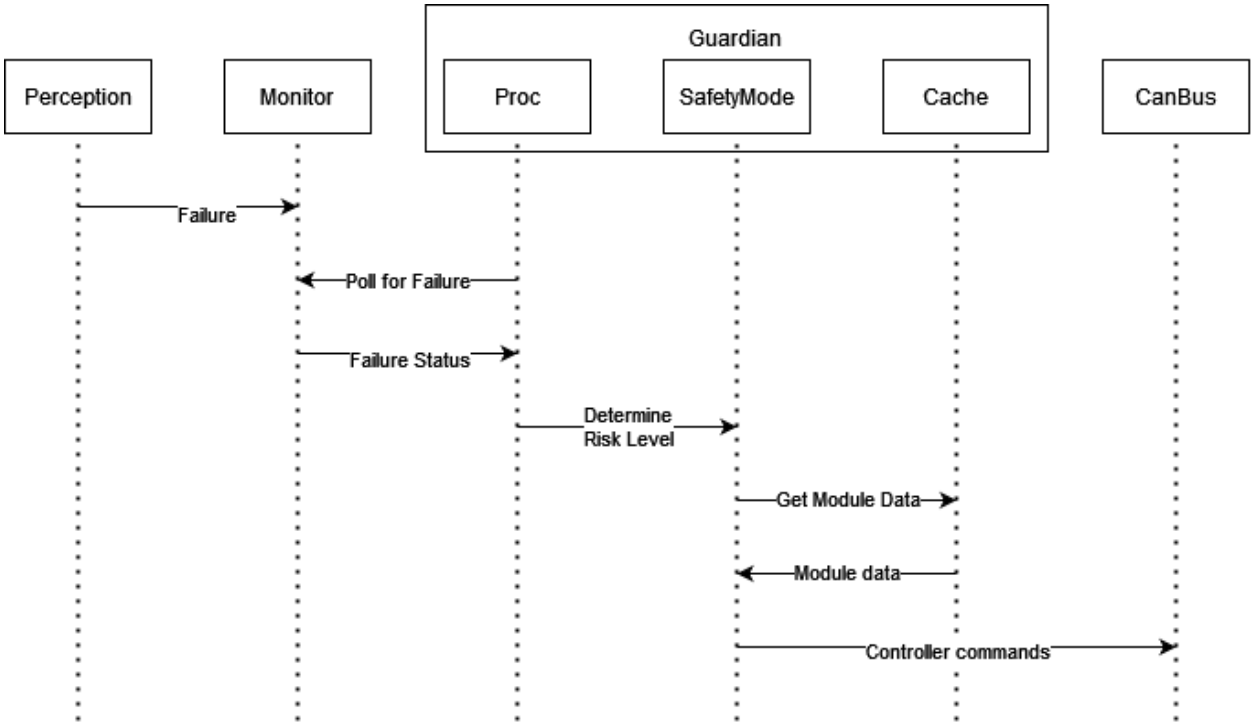

Figure 5. Sequence diagram of Medium-Risk use case

In this use case, the perception module has failed and the guardian will enact a medium-risk plan for bringing the car to a stop safely. When the perception module fails, the monitor will tell the guardian the next time the monitor gets polled. The SafetyMode submodule determines that the risk level is medium. It then gets the cache and plans a route for the car to go. It sends controller commands to the CanBus to enact that plan.

Potential risks

Lack of safety is the most important potential risk there is to discuss in the autonomous vehicle domain. As the new enhancement is an additional feature to help passengers inside the car to be safely routed to a stop, the whole process needs to have high accuracy. The vehicle should have accurate cached data in order to prevent an unsafe stoppage that defeats the purpose of this enhancement.

Performance could also become a potential risk that could ultimately lose the level of safety of the new enhancement. For example, if the Proc subcomponent fails to check module failures fast enough, it lowers the overall level of safety of the new enhancement as the Guardian

module may already be too late to calculate the risk level for a hard brake if the situation needs immediate attention.

Availability is also important to pay attention to because the failure of the Guardian subsystem itself could create a potential risk as the failure of the Guardian subsystem loses the overall responsibility of constant checking of other subsystem failures.

Lastly, lack of attention to Scalability can become a potential risk as the new enhancement deals with constant data transfers between cached memory inside the Guardian subsystem. There's a possibility of bottlenecking of the hardware dealing with too much data with the current hardware we are currently using before the enhancement. This could end up with the subsystem not being able to provide the cached real-time data which defeats the purpose of the new enhancement.

Test plan

Test cases for the new enhancement can be separated into three parts of the Safety Mode. As mentioned previously, the new enhancement is easily testable as it can be separated into subsections with different levels of risk. Each risk level has a specifically impacted subsystem of the Apollo architecture so it is easy to test each case of subsystem failure with the provided test plan.

Low-Risk Use Case

When subsystems are not directly involved in the main function of autonomous driving, the Guardian activates a low-risk stop that creates a safe stopping route. This use case can be tested by having sub-test cases for each subsystem failure which includes Task Manager, Dreamview, HD Map, and Routing subsystems. The goal of the Low-Risk testing is to make sure that the autonomous vehicle safely routes itself on the side of the road to fully stop when there are failures of any of the listed subsystems, using key subsystems that are still working.

Medium-Risk Use Case

Medium Risk stop occurs when real-time data isn't available from the following subsystems: Storytelling, Localization, Perception, and Prediction. The Guardian module sends a command to the Planning module to create a new safe stopping route using the cached real-time data from 5-10 seconds before the failure of mentioned subsystems. The main goal of Medium-Risk testing is to test both the accuracy and safety of the cached data usage which ultimately stops the car safely on the side of the road.

High-Risk Use Case

High Risk is when the autonomous vehicle is unable to create a trajectory due to related subsystems (Planning, CanBus, Monitor, Control) failure, or sensor/hardware failure, the Guardian module brings the vehicle to an immediate stop. This is very much like the hard brake function that the original Guardian module had.

Lessons learned & Limitations

Limitations

As this is a new concept that we came up with, there were no resources that we could refer to. This meant that we had to figure out the architectural details and implementations ourselves.

There is no guarantee that the cache submodule would work without more resources including the hardware and autonomous vehicle for further testing.

Lessons Learned

We learned to use the SAAM analysis approach to actually implement non-functional requirements to our enhancement and compare it to come up with a better solution. We also got to be in a different point of view for each stakeholder to find relations with non-functional requirements.

As mentioned previously in the limitations section, coming up with a new enhancement that upgraded the overall subsystem was difficult without the help of outside resources. However, it was a great learning experience for us to figure out and manipulate new sub-components to enhance the overall system.

Conclusion

Through the discussion on what part of Apollo we should focus on, we decided to go with the enhancement on the Guardian module.

We suggested two approaches to implementing the enhancement. Approach A is rejected because of the redundancy of failure handlers. In approach B, different situations are categorized into three levels, which are enough for cars to react to emergencies. The Guardian module basically makes a choice among three options when failure messages are delivered. With the cache data in the Guardian module, it can also commend the Planning and Storytelling modules to generate a trajectory leading the car to a safe place. Via the SAAM analysis, approach B also stands out from the aspect of NFRs.

Adopting approach B to the system brings new dependencies between the Guardian and other modules. To identify the level of dangers and interact with other modules, the Guardian also introduces several new components to itself, like SafetyMode which classify failure level.

The report further explains the categorization of three levels of risks. Low risk will guide cars to a safe stop based on the stored knowledge. Medium risk occurs when the Guardian has to inform the Planning module to generate a safe trajectory without any useful scenarios in the Storytelling module. At last, high risk occurs when no modules function in a proper way, which will stop the car immediately.

Data Dictionary

Architecture: High-level structure and organization of subsystems in software.

Subsystem/Module/Component: A self-contained system within the overall architecture. These three words are used interchangeably.

DreamView: A web application that helps developers visualize the output of other relevant autonomous driving modules.

HD Map: A high-precision map loader interface.

Cache: Hardware or software component that stores data so that future requests for that data can be served faster.

Naming Conventions

Pub-Sub - Publish and Subscribe Architecture

CanBus - Controlled Area Network Bus

SAAM - Software Architecture Analysis Method

NFR - Non-Functional Requirements

References

[1] GitHub repository of Apollo Auto
https://github.com/ApolloAuto/apollo

[2] Concrete Architecture (Assignment 2)
https://aleksjug.github.io/artemis/

[3] Ciraci, Selim & van den Broek, Pim. (2006). Evolvability as a quality attribute of software architectures. Journal of Physics: Conference Series.
https://ris.utwente.nl/ws/files/5397018/Ciraci06evolvability.pdf