CISC 332
A3: Proposed Architectural Enhancement
4/11/2022

Marc Brasoveanu	18mtb5@queensu.ca
Andrew Wilker	18ajw15@queensu.ca
Nathan Perriman	17nmp2@queensu.ca
Theo Raptis	17tjr5@queensu.ca
John Mahone	21jgm18@queensu.ca

Table of Contents

Abstract

This report focuses on a proposed enhancement of the current architecture for the Apollo Autonomous Driving library. The enhancement that we have proposed is a data propagation system which allows all vehicles running the Apollo Autonomous driving software to communicate with each other in real time. This will be achieved through the creation of a new ‘Communication’ module along with modifications to existing modules.

Some of the key findings from this report were, reiterating the current state of the concrete architecture where we identified 12 different modules which worked together in a publish/subscribe architecture to facilitate the autonomous operation of a vehicle. Some of the interactions between existing modules and our proposed enhancement include interactions between the perception, routing, map, and planning modules. The mentioned modules are important in the interaction with our proposed enhancement since they all have to do with information regarding vehicle status, information about its surroundings and transmitting this information. Two use cases that we focused on to demonstrate the usefulness of our architectural enhancement were the detection and communication of a traffic jam as well as the coordination of vehicles at a red light.

Some of the important non-functional requirements regarding the enhancement include maintainability, evolvability, testability, and performance. Some of the alternative implementations for realising our real time information propagation system include, preserving the current architecture and just integrating the enhancement by making modifications to all the existing affected modules as opposed to creating a new ‘Communication” module to act as the control centre for the new enhancement. The creation of a ‘Communication’ module was determined to be the best course of action. Some of the main stakeholders include, the development team, the users and any members of the programming community who look to modify the open-source code to their needs. These stakeholders should be concerned with NFRs such as security, performance, maintainability, and scalability.

Some of the modules which will be affected by our enhancement include the map, planning, and routing modules. Testing for the first use case would include creating an artificial traffic jam and sending a test car towards it while testing for the second use case would include vehicles designed to collide multiple times at a busy traffic light. Risks from improper implementation of our enhancement include serious injury or death from lack of testing and lack-lustre communication.

Some of the limitations include unexpected dependencies and the feasibility of our proposed enhancement due to technology maturity and hardware compatibility.

Introduction and Overview

The purpose of this report is to propose an architectural enhancement to the current Apollo architecture. We have chosen to propose a real-time information propagation system which allows Apollo vehicles to communicate with each other in real time to send information regarding the current status of surrounding vehicles and environmental factors.

The report starts with a mention of the value of our proposed enhancement and then is followed by the current state of the concrete architecture, discussing the use of the publish/subscribe style to enable the high degree of concurrency between the 12 different modules listed. These modules work together to collect information about the current scenario of a vehicle which includes its location, surroundings, and destination.

Next some interactions between modules and the proposed enhancement are discussed. The perception module which is responsible for gathering information regarding the vehicle's surroundings, the routing module which plans the vehicle's route, the map module which deals with the vehicle's current location, and the planning module which makes movement decisions based on all of this information are all affected by the proposed information propagation system. All of these modules have to do with the vehicle being able to accurately and safely engage in autonomous driving and furthermore affect the real-time information being exchanged between vehicles in our proposed enhancement.

The next section of the report deals with the use cases that demonstrate the usefulness of our architectural enhancement along with sequence diagrams associated with those use cases. The first use case is if one vehicle detects a traffic jam and notifies other self-driving vehicles so that they can coordinate and determine alternative paths though the routing modules. The second use case is coordinating the efficient flow of vehicles at a traffic light so that there is minimal delay between cars beginning to move once the light turns green.

Some of the non-functional requirements which should be a concern for stakeholders include maintainability which is affected by the need for new testing modules, evolvability should not be affected, testability which is affected by new functionality surrounding the proposed use cases and the need for new tests to ensure proper behaviour. Performance of the entire system will be affected by the bottleneck of the performance of the communication module.

There are two different implementations of the information propagation module that we considered. The first being to keep the existing top-level architecture intact and spread the new feature among each of the modules that it affects. This would likely decrease the performance of the system since more information is now being sent through the same architecture. Also security concerns are present with the integrity of network information being intercepted. Scalability is also a concern for developers. The second implementation would be to create a new Communication module which would control the information propagation process. The maintainability and scalability of this approach would be much better than the first approach.

Performance would also likely be better due to a publishing style similar to the storyteller module. We believe the second implementation is the preferred choice.

The architecture will be affected in certain key modules such as the map, routing, and planning modules. The map module needs to be changed to handle the communication subsystem and must be able to output map data for it. The routing module will be affected because it will need to accept new information from the map sent by other cars to use in the route generation. The planning module would need changes to deal with how to handle movement associated with traffic light data in the second use case. The addition of the communication subsystem is also an obvious architectural change to be considered.

Testing would need to be added for the new functionality. Specifically the two use cases need to be addressed with unique test cases. First the first use case an artificial traffic jam would need to be created and tested with cars accepting that new information and making sure they re-route accordingly. For the second use case traffic light information would need to be simulated and the communication between vehicles at the light needs to be observed.

Risks associated with the enhancement include serious injury and death from the improper movement of the vehicle due to inaccurate information or improper communication with other vehicles.

The report concludes with naming conventions, a data dictionary as well as a summary of the conclusions of the report. Finally some limitations are listed such as whether or not it is possible for our proposed enhancement to be realistically implemented in the current architecture due to compatibility concerns with hardware or feasibility of the technology. Also unexpected dependencies may pose a concern. We learned how to undergo the SAAM process and understood the importance of knowing each subsystem in detail along with a holistic view of the system.

Value of Enhancement

The value of a real-time information propagation system is quite substantial. The ability for vehicles to communicate with each other in real-time allows for improvements in traffic flow for all parties involved. The two use cases that we discuss, the first being detecting a traffic collision and letting other vehicles on the network know in order to set a more optimal route, and the second being increasing traffic flow at a traffic light by reducing the delay between stopping for a red light and moving for a green light. In both of these cases the valuable time of drivers will be saved and over the course of months and years tens of hours of time per driver can be recouped. Potential safety benefits also exist in rerouting vehicles around a collision so that a dangerous environment is not created for those involved in the collision as well as oncoming vehicles which aim to avoid the debris.

Current State

In our previous report, we worked to uncover the concrete architecture of the Apollo system to gain a greater understanding of the software as it currently exists. We use the work completed in the previous report to first provide a brief overview of the current state of the Apollo software.

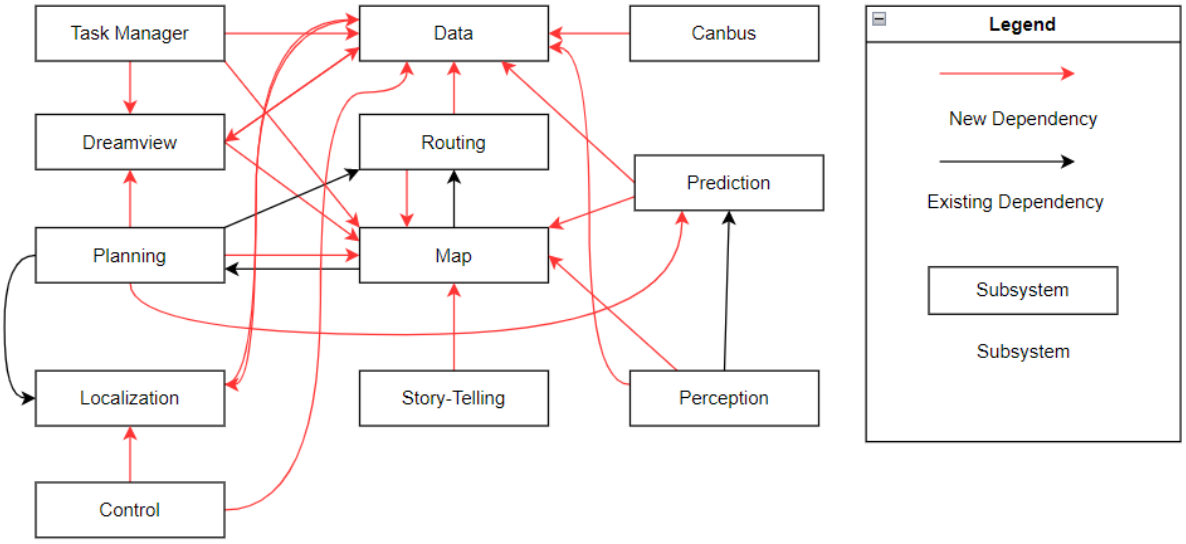


Figure 1 - Derived Concrete Architecture

The system in its current state consists of 12 different modules, each fulfilling a unique role. The architecture itself follows the publish/subscribe style in which modules send information to one another by “publishing” messages to a broadcasting system/bus, and receive information by “subscribing” to topics of interest so that they receive only relevant messages. The architecture of Apollo is also highly concurrent, owing to the dynamic nature of the autonomous driving problem domain, which necessitates the ability to be able to quickly perform multiple independent calculations while keeping response time to a minimum. The 12 modules of Apollo work to meet requirements for understanding the current scenario of a vehicle (location, surroundings, destination) and modifying and adjusting current plans to operate the vehicle completely autonomously.

Interactions

Modules important to our proposed enhancement include the perception, routing, map and planning modules. The perception module takes on the responsibility of gathering information about the vehicle’s surroundings through the use of camera, radar and LiDAR sensors. The routing module plans the vehicle’s route based upon mapping information, as well as the current location of the vehicle and its desired destination. The map module handles information regarding the current location of the vehicle, as well as the generation of mapping information using the real-time

location of the vehicle. The planning module plays a key role in the whole architecture by determining the current scenario of the car based upon a set of built-in scenarios, and using this information to guide the vehicle's actions. Because our enhancement aims to allow vehicles to communicate real-time information about their whereabouts and route to one another, it must interact with all of the aforementioned modules. In order for any information regarding vehicle status to be useful, it must be accompanied by information regarding the location of any given vehicle, as well as information about the vehicle's surroundings. This will require interaction with both the perception and map modules. In addition, the module will need to interact with the map, routing and planning modules so that information can be provided to allow the planning module to evaluate the vehicle's current status and route plans. The above dependencies are necessary in order for the vehicle to be able to transmit information to other vehicles, as well as to be able to act on information received from other vehicles.

Use Cases & Sequence Diagrams

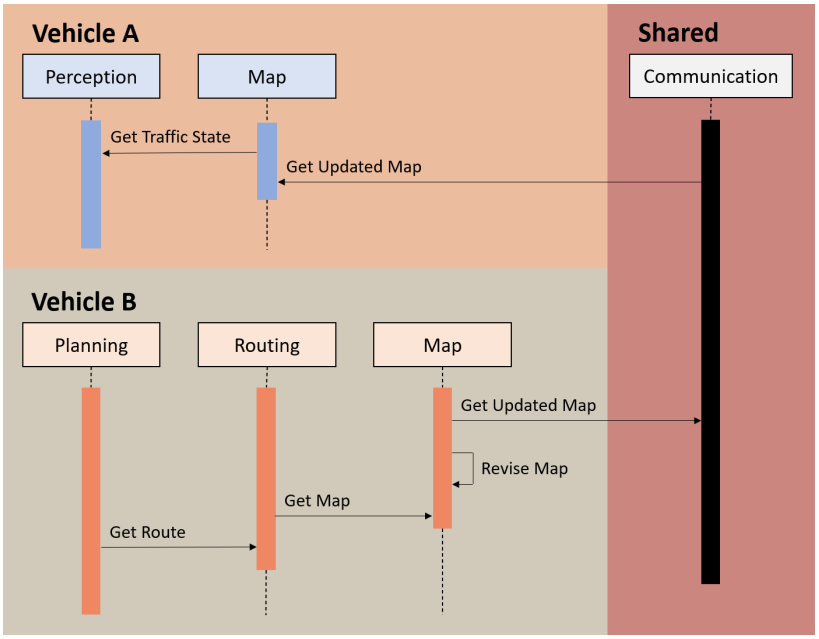
Here are two use cases that demonstrate the usefulness of our architectural enhancement. Our first use case is when an automobile detects a traffic jam that other self-driving automobiles are not aware of.The seconds is a set of self-driving automobiles coordinating at a red light.

Use Case 1:

An automobile detects a traffic jam at some location and wants to send a signal to other vehicles in the network so that they can efficiently determine a new plan. The automobile will have the capability to do this via the use of our new communication module. For this use case, we identify the detector automobile as 'A' and the receiver automobile as 'B'.

Automobile A will detect a traffic jam using its perception module and send this data to the map module. Automobile A will continue to operate as normal, except for the fact that it will send this new map information to our communication module. This module is shared by the network of Apollo vehicles. From the communication module, automobile B will send the updated map information to its map module. From here, automobile B will operate with the newfound knowledge now contained in the map module; it will send the updated map to the routing module, which will send a route to the planning module, causing automobile B to have a new plan that avoids the traffic.

Sequence Diagram:

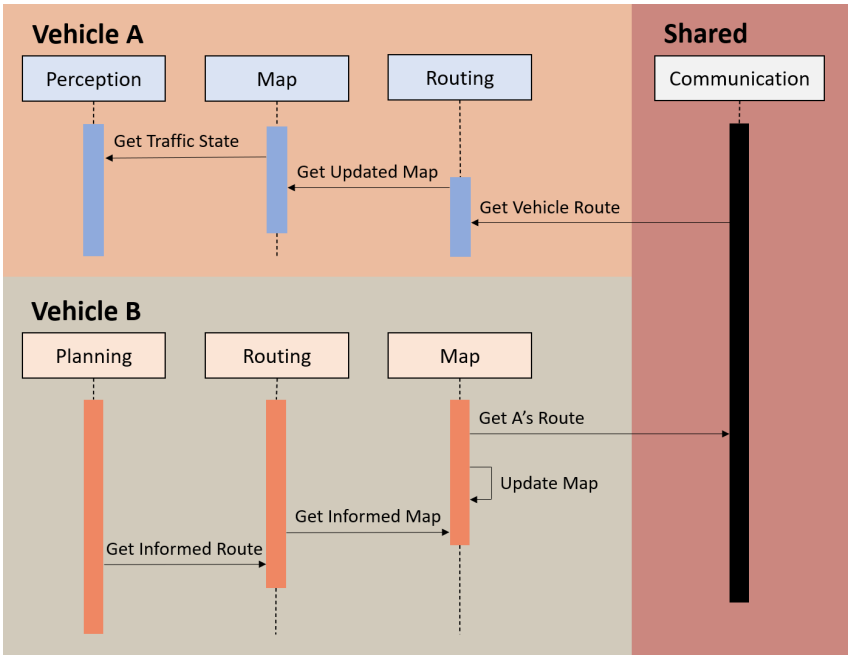


Use Case 2:

Regularly, two cars cannot move synchronously due to the risk of human error in accordance with reaction time. This is obviously suboptimal, as one automobile must wait for the automobile in front of it to move, delaying the vehicle. This concept is what motivates this use case; automobile B is behind automobile A, automobile A coordinates with automobile B such that they both have an unobstructed plan that incorporates both vehicles when the stop light turns green.

Like the previous use case, Automobile A will detect that the light has turned green by using its perception module and send this data to the map module. Automobile A will resume its routing with the new data from the updated map module, stating that the traffic light is green. As a new operation, the automobile sends this new route to the communication module. From the communication module, automobile B receives automobile A's new route, which is catalogued by the map module. Finally, the updated map, which now includes the route that automobile A is taking, is fed into automobile B's routing module. Finally, a new route is determined and sent to the planning module, causing Automobile B to have an updated, more efficient plan, based on the information that automobile A has sent automobile B.

Sequence Diagram:



Non-Functional Requirements

Maintainability

The enhancements should have a small effect on the system's maintainability. Any automated testing that Apollo employs on its current system would still be easily done with the addition of the communication module. Since the communication module is a new addition to the system, a new set of maintenance procedures would have to be designed by Apollo. These might include new automated testing for the communication module and keeping logs of communications made to ensure everything is operating correctly

Evolvability

This enhancement should not negatively impact the evolvability of other modules in the system, because the communication module does not require other modules to behave in a particular way. The communication module only passes around data that is already present. Once set up, the communication module would have the ability to be evolved by the programmers quite easily; if we find new information that we want to propagate to our network of vehicles, we can quite easily grab that information from a pre-existing module. No new patterns would be required.

Testability

The testability of this enhancement could be somewhat inconsistent. In creating a test environment, it might be somewhat difficult to determine what the new behaviour should be, if any. For example, in use case 1, we might expect the vehicle to reroute around traffic, and determine that there is a bug when the car heads into traffic. This may be a false flag, however, if the vehicle found that heading into traffic would still be faster than a new route. It might indeed be quite nebulus for software engineers to determine if the system is behaving correctly.

Performance

Because the communication module is not doing any operations, just requesting and sending data, this enhancement should have a negligible impact on the system's performance. The performance of the module itself has a dependency on the quality of the network it is sending its data through.

Alternative Implementations

To implement this information propagation into the Apollo system, there are a number of different approaches that can be taken. From a software point of view, there are two distinct ways that the improvement can be added. The first is to preserve the current working architecture of the Apollo system and integrate each of the features of this enhancement into the existing structure. The second approach would instead be to modify the architecture, bolstering it with the addition of a new 'Communication' module that can exclusively handle the new demands of the improvement. To determine the ideal approach towards implementing this improvement into Apollo, we first need to analyse how these improvements are going to impact those involved in the project.

For the enhancement of information propagation, the main stakeholders in this improvement are the development team, the users and any members of the programming community who look to modify the open-source code to their needs. For these stakeholders, some of the most important NFRs will be the security, performance, maintainability, and scalability. Now that we know which NFRs are of the most importance for this improvement, we can undergo a SAAM analysis on each of these alternate methods of integration, in order to find which of these methods is more likely to be effective at implementing the improvement.

SAAM Analysis

Our first proposed strategy for implementation of the information propagation system is one in which we would maintain the existing system architecture and

implement the additional features of the system into the existing modules. The components of this new feature would be spread among each of the modules that it would affect, and the top-level concrete architecture would remain virtually unaffected after the improvement is implemented. This enhancement seems appealing to the developers of Apollo, as it doesn't require any single large software alteration. Instead, the components of this feature are inserted where they're relevant. Seeing that this method of integration means there's no additional modules to communicate with, it's easy to assume the performance of this implementation could be slightly decreased, as calls through this implementation would slow the entire module in order to send or retrieve external data to process. However, from a security standpoint this method introduces some serious hazards, as we can assume that any kind of breach in the network security of the vehicle would give any external parties access to the entirety of any module that performs network communication, which could leave the vehicle's operator extremely vulnerable. The other major concerns are with the maintainability and scalability of the software. If any programmers, either in the development team or an outside party, wants to modify or improve the communication feature in any way, they will be forced to shift through each of the modules to locate which parts are in control of the communication of data.

However, our second proposed strategy of this implementation is to instead create an entirely new module, which would instead be in control of the entire information propagation process. This module would mean a great shift in the concrete architecture of Apollo, as this new module would need a number of dependencies in order to function properly. This module would be a bit more difficult to implement for the Apollo team, as they would instead be required to remap the software and allow the module to receive data from a number of other modules. From a performance point of view, this method likely wouldn't affect the existing performance of the system due to a publishing style similar to that of the storyteller module. In the event that there's a fault and this module slows or stops in any way, the other modules can resume their functions without interruption. Lastly, the affect this style of integration would have on the maintainability and scalability of the improvement would be hugely positive. With all the features of this system located in one central location, any bugs to remove or any features to add are very easy to locate.

After an in-depth comparison of the two integration methods, it is easy to see that although the method of integrating into the existing architecture may be slightly easier, there are a lot more benefits when considering the alternative of implementing an entirely new method to handle these changes instead. That is why the method our group decided to choose in order to propose this new enhancement is to create a new module 'Communication' in order to incorporate the features this would require.

Effects on Architecture

Our suggested features would impact a number of the subsystems and modules that we discussed in the conceptual architecture. The changes and interactions that need to be added are described for each affected module below:

Map

The map module needs to be changed to handle the new features that are being implemented. In the first use case, it must now account for the communication subsystem that we have created and thus must be able to output the map data to it. In the second use case, the map module will provide information as it currently does to the routing module.

In both of our use cases, the map module must also accept information that it gets back from the communication module. It will then factor this additional mapping information into its mapping processes and build a more detailed 3D map that will be used for routing and planning that includes information gained from other vehicles and their planned routes.

Routing

The routing module will need to be able to accept the new information from the map subsystem and use it in route generation. After taking in the new map information along with the requested route, it will now have to factor in traffic from the first use case and if that will affect the cost of a route it creates. It will need to update the available series of roads it can take and the accompanying turns when creating the route through.

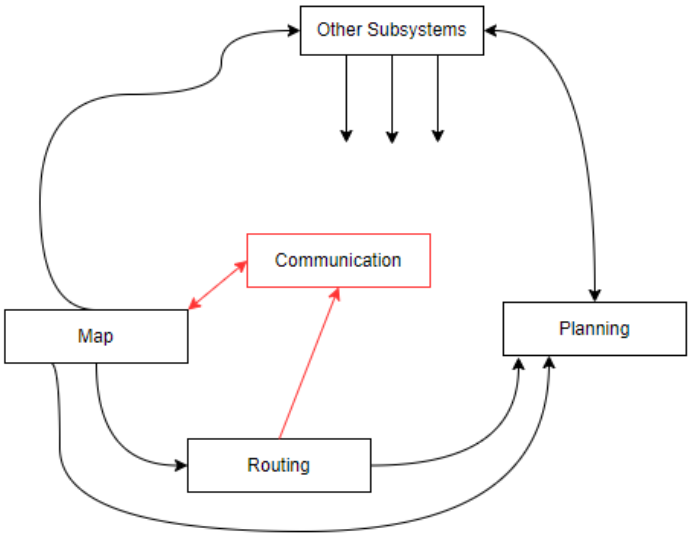
In the routing module, we would need to alter the files in the *topo_creator* directory for the new input it can receive from mapping for a traffic jam.

Planning

The planning module needs some minor changes so that it can use the new use cases we have proposed. The first one is to expand how it deals with traffic light setoff so that it can account for the second use case of our feature. Other than that, the planning is based off of the maps that have factored in the other vehicles traffic readings and vehicle routes.

The *traffic_light* directory in the planning module would be changed to account for using the first vehicle alert that you can start moving.

In addition to the above changes, there would also be an entirely new subsystem implemented within the architecture. This subsystem would be the **Communication Subsystem**. For the first use case it retrieves information about the map details related to a traffic jam and allows these details to be retrieved by other instances of the Apollo software in use so that they can factor into the further maps being generated. In the second use case, the communication subsystem sends a notification that other cars located behind the current one will be moving as well.



Condensed conceptual architecture showing how the communication subsystem would be implemented.

Testing

The improvement that we wish to introduce to the software would require extensive testing before it could be released because it could endanger lives due to the vehicles that use the software. The testing should involve extensive testing digitally before being used in various real world test scenarios. For example, it would need to test the traffic feature in a private testing facility where any potential accidents can be safely contained.

Tests for the first usage case would involve creating an artificial traffic jam. Once that is done, we would send a test car towards it and check if a signal is sent from the car, then received by any of the receivers in the testing cars. Once we have tested the sending and receiving feature, then we can begin testing in having the Apollo software determine a new direction that avoids the current traffic jam. This could be done by analyzing the route selected once the traffic jam has been detected and comparing it with the possible options to see if the car selected the best one based on the current traffic jam data.

The second usage case would require different tests using vehicles designed so that they can collide multiple times. This feature involves multiple machines communicating and moving at the same time, and thus there is a high chance that during testing there would be multiple collisions which we would have to account for. Before this feature could be released into any real world scenarios, we would have to ensure that it passes rigorous tests.

Once we have completed these tests, we could begin deploying it into areas where autonomous cars are already being tested with these enhancements. These

real world scenarios would require observations from a handler who can monitor the signals between the cars and ensure that any problems are handled.

Risks

The main risk comes from our second feature that we wish to implement, the ability for cars to communicate and navigate an intersection of red lights. Even after extensive testing, there is still a notable possibility of the software encountering a situation that our tests didn't cover and a vehicle collision occurs. This fallout from such a feature could range from low level damage, up to serious injury or death, which means that this second feature must be rigorously tested as described in the previous section.

Naming Conventions

NFR - Non-Functional Requirement, a criteria that is important in the operation of a system, but not necessary to run.
SAAM - Software Architecture Analysis Method, used to evaluate the architecture of a piece of software.

Data Dictionary

Pub-Sub/Publish-Subscribe – Architecture style where modules subscribe to a publisher module
Concurrency – Multiple items are run at the same time
Localization – Module that lets the automobile know where it is
Map – HD map module
Perception – Module that detects objects in images
Planning – Module that plans the auto's motion
Prediction – Module that predicts what is happening in its environment
Routing – Module that makes routes
Storytelling – Manages other modules
Sequence Diagram – Diagram that shows the sequence of events over its modules

Conclusions

In conclusion some of the key findings were that the current concrete architecture uses a publish/subscribe style to facilitate a high degree of asynchronous communication between 12 main modules. Some of the modules which will interact with our proposed enhancement of a real-time information propagation system include the map, routing, perception, and planning modules. Two use cases of our enhancement include detecting and sending information about a traffic jam along with coordinating traffic flow around vehicles stopped at traffic

lights. Some NFRs that stakeholders should be concerned with include, maintainability, testability, performance, and evolvability. The two implementation methods we considered were creating a new communications module vs keeping the existing architecture and just adding more functionality to existing modules. We determined the addition of a communications module to be a better alternative. New tests would have to be written for both use cases and risks would need to be evaluated to the complexity of the communication of information. Some of the limitations we considered were the feasibility of our proposal due to hardware compatibility and technological maturity.

Limitations and Lessons Learned

This report comes with some limitations. While we have made our best attempt at determining how our enhancement would fit in with the system, it is impossible to say whether our assessment of the required changes, our testing plan and our risk analysis are completely correct without actually attempting to implement our changes. The analysis done within this report is analogous to a conceptual architecture; we have identified on a high level the changes that need to be made and drafted out a likely idealised plan to implement them. However, in reality it is much more likely that changes would need to be made to the plans we have created in order to fully implement our enhancement, and also likely that additional unexpected dependencies may be added during the development process that were not anticipated in this report.

In addition, it is impossible for us to say whether our proposed enhancement is feasible or not without attempting to implement it. During a real-world development process, it is possible that the proposed changes may be found out to be impossible or impractical with current hardware. It is also possible that changes could be scrapped if the amount of work required to implement these changes is not worth the benefit to stakeholders. Stakeholder priorities could also change, and the enhancement may no longer be desirable.

As a team, we feel that the process of investigating how to implement such an enhancement has given us a much better understanding of the SAAM process, as well as the steps required to build a testing plan and risk assessment. Understanding how to incorporate our new changes also allowed us a chance to better understand the Apollo architecture. We had to make sure we completely understood the purpose of each component in order to assess which components would be affected by our enhancement. Once we identified these components, we had to develop a low level understanding of them in order to understand how they would be affected.

References

1. https://apollo.auto/developer.html