CISC 322
Assignment 3 Report
Apollo: Autonomous Driving Vehicles
Architectural Enhancement Proposal
Friday, April 8, 2022

Group 1: Hextech

Benjamin Hui (Ben) (He/him) (18bh13@queensu.ca #20148554)
Fuwei Zhuang (Elina) (She/her) (19fz2@queensu.ca #20189056)
E Ching Kho (Noon) (He/him) (17eck3@queensu.ca #20118077)
Zewen Zheng (Zelvin) (He/him) (18zz114@queensu.ca #20150673)
Yixin Su (Allen) (He/him) (17ys114@queensu.ca #20108833)
Ruiyang Su (Amelia) (She/her) (18rs60@queensu.ca #20151860)

1.0 Abstract

Following the analysis of the conceptual and concrete architecture of the Apollo system, our team has developed a potential enhancement that can be added. Its name is Family Share, and its intended use is for groups of people in the form of a "family" to be able to interact with each other. The two uses discussed in the report were the sharing of positions within the group and the sharing of destination and estimated arrival times. This enhancement will introduce a new module named Sharer and will communicate with localization, planning, routing, dreamview and the monitor modules. A SAAM analysis [1] was also conducted to analyze the impact and proper documentation of the proposed enhancement. Including potential stakeholders, implementation NFRs, concurrency, and testing. We also go into depth about its impacts on the high-level and low-level architecture and how the respective modules interact and affect. We also discuss two use cases to illustrate the implementations of Family Share.

2.0 Introduction

The Apollo system already features autonomous driving capabilities, and this enhancement Family Share aims to add to the functionality and accessibility of the system. Family Share will allow users to communicate more efficiently without using external devices such as smartphones. By entering the destination on the Apollo system, users will then have an option to share their destination in their group. This enhancement function can be seen as a "group chat" separated into two different uses. One is the sharing of each other's positions. For example, users in a group/family will be able to share their current location, and other users will be able to see their current location on their maps. The other use is the sharing of a destination and their arrival times. This will allow groups of people to plan their travels easily and plan ahead and make sure everyone arrives at the destination on time. Below we will discuss the overview, functional and non-functional requirements, a SAAM analysis, the enhancement's effects on the conceptual architecture, and two use cases with sequence diagrams.

3.0 Proposed Enhancement

3.1 Overview

Our proposed enhancement is called ‘Family Share’ which aims to connect families or groups of people. The system has the following functions:

- The *Family Share System* will allow users to share their current locations while viewing other members in the same channel. Positions of vehicles will be shown on the map display.
- Another function will be to share a destination with an estimated arrival time that can be sent to other members within the channel. Members will be able to see each other’s arrival times and start heading towards the destination through their own routes.

For each function, users have the ability to choose to disable the functions due to privacy concerns.

As the *Family Share System* is designed and curated for groups of people, it aims to improve the user experience and expand the functions of the Apollo system.

Sharing locations will allow groups of people to follow each other when traveling together and can also allow them to see if anyone has crashed or gone off to the wrong route. Our *Family Share System* is also devoted to improving the experience of multiple autonomous car driving situations. For example, at a Chinese wedding, it is a tradition that the groom brings the groomsmen and his other family friends to the location of the bride. Due to a large number of people, this required multiple vehicles to travel to the same destination. The function of Family Sharer will be able to plan and attempt to avoid situations such as delays, traffic jams, and problems with punctuality. It will allow people to try to plan ahead and arrive at their destinations at the same time.

One thing to note is that in Apollo, there is a component called v2x and it contains two submodules: Roadside unit (RSU) and On-board unit (OBU). OBU records the car's status and sends it to a server through grpc protocol, but the purpose for that is to effectively alleviate traffic congestion, reduce traffic pollution, and improve traffic safety. It is similar to a function of our enhancement but with a different purpose as ours is to create a vehicle to vehicle communication and improve the driving experience between a group of users. Thus, improving the Apollo system.

3.2 Architectural Styles

The architectural style of Apollo could have some changes. Apollo's architectural style will add Client/Server to Pub-Sub by introducing Sharer. The Sharer module (with a submodule called Networker) in each vehicle acts as a client subscribing to the Family Channel, a namespace stored in Apollo's Cloud Service System. This allows straightforward distribution of data, transparency of location, and easy to upgrade new clients.
Alternatively, the main architectural style of Apollo can become a hybrid by combining the Pub-Sub style and Peer-to-Peer style. As a result, the system will be scalable since there is no central server. In addition, the user will have more privacy, and it will be easy to add new family members or upgrade existing members to the *Family Share System*. However, there will be no guaranteed response from other vehicles, and this could be due to down, delayed, or hostile.

4.0 SAAM Analysis

4.1 Implementation Analysis

4.1.1 Implementation Overview

In order to implement the stated new features, we came up with two implementations, both based on the original Pub-Sub architectural style, but using different architectural styles to implement our *Family Share System*.

Implementation I:
The first implementation we labeled is connecting vehicles using the Client/Server architectural style. As the following figure (**Figure 1.**) illustrates, there is a centralized cloud server for this implementation. Each vehicle acts as a client in the *Family Share System*. It calls on the services provided by the cloud server to get information from other vehicles, for example, locations, destinations, estimated arrival time, etc. The vehicles should also update their current data to the central server for other vehicles in the channel to retrieve.

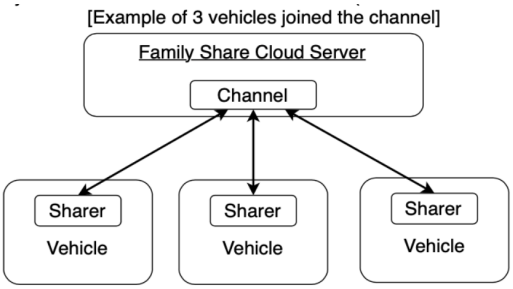

Figure 1. Implementation I: Car-Network-Level Architecture (Pub-Sub + Client/Server)

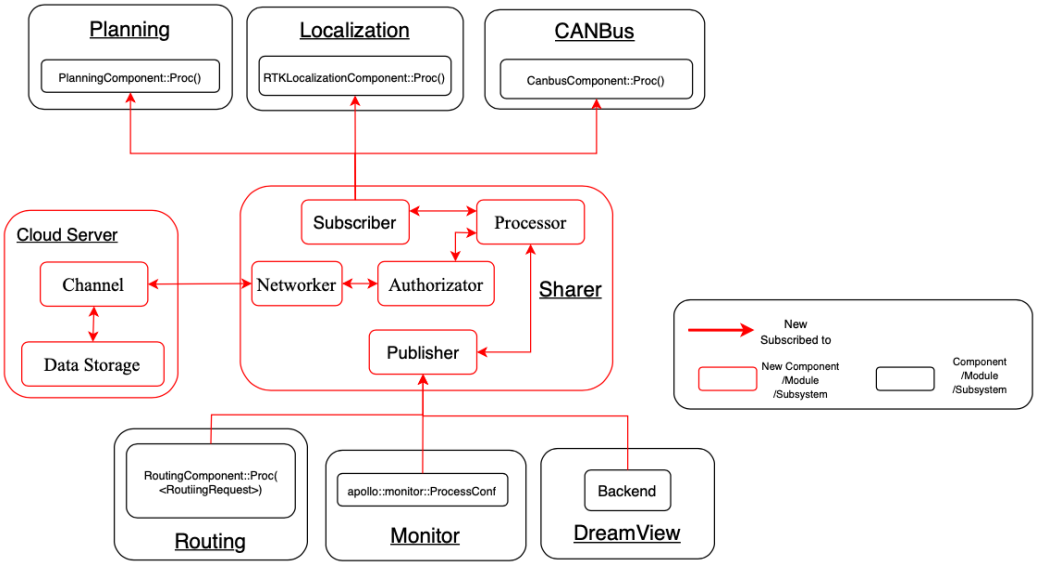

Figure 2. Implementation I of feature into conceptual architecture subsystems

Implementation II (Alternative):
The second implementation is to design a Peer-to-Peer system for vehicles connection. In this implementation, every vehicle has direct access to other vehicles in the family channel without a central server. To this end, we will need to create two additional components compared to implementation I. First, we need to have a Registry component to record the IP addresses of the vehicles in the same family channel for identification for data extraction. Second, each vehicle should have a data storage component to store its and others' information, which must be updated continuously.

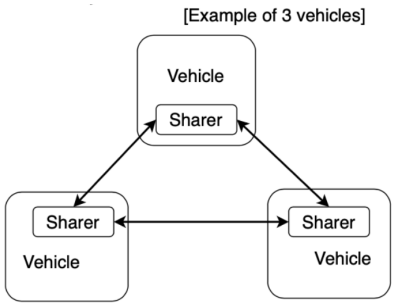

Figure 3. Implementation II: Car-Network-Level Architecture (Pub-Sub + Peer-to-Peer)

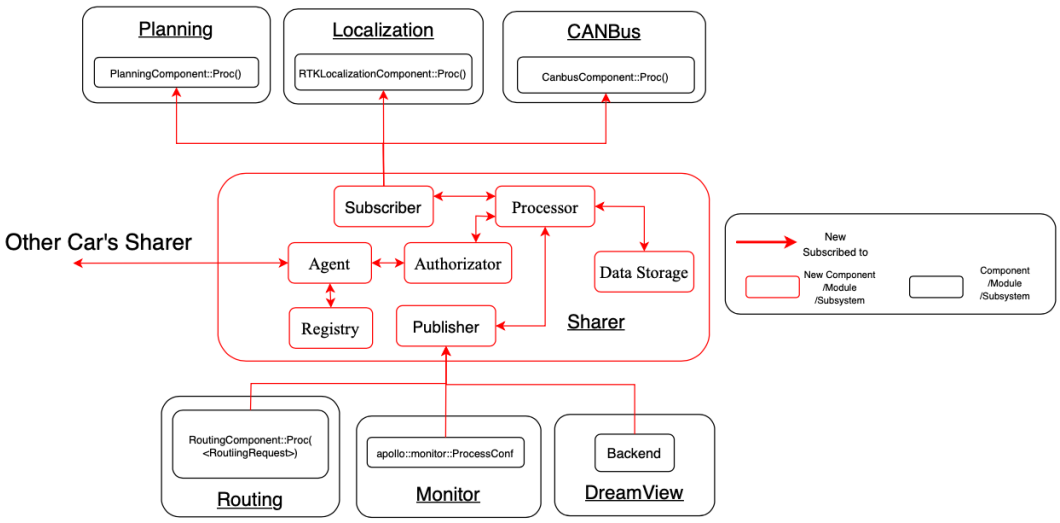

Figure 4. Implementation II of feature into conceptual architecture subsystems

4.1.2 Implementation NFR Analysis

NFRs	**Implementation I**	**Implementation II**
Performance	Medium performance. The user can receive information about all other cars within a single data request to the family share channel. Through the connection of a vehicle to the	High performance. Since there is no central server, there is no single point of failure. The vehicle can directly fetch data from another vehicle and

	channel then the channel receiving data from other vehicles may be delayed.	no need to wait for Cloud to transmit the data.
Usability	High usability. Users have the right to choose whether to use the feature or not, which requires authorization. When using the feature, users can authorize the amount of data they are delivering to the channel.	High usability. Users may choose to use the feature by permitting other vehicles for data extraction and vice versa.
Availability	High availability. The cloud server should be active 99% of the time and users can choose whenever to connect to the channel.	Low availability. There is no guarantee of response and connection to other vehicles.
Maintainability	High maintainability. Server status can be checked to ensure that the server is working properly, and should prevent cyberattacks or damage. Also, the Monitor component needs to inspect the condition of Sharer when running and the internet should be stable.	Low maintainability. We need to ensure that every vehicle is working properly, and connections should be stable.
Testability	High testability. The testability of the function is easy to check. We can compare the information sent by the vehicle and the information received in the channel is accurate and consistent.	Low testability. When there are a lot of vehicles, it is tedious to keep track of one vehicle's data sent to all other vehicles and where the problem in transmission occured.
Security	Medium security. Data disclosure and leaks are possible when the Cloud Server is being attacked.	High security. Every vehicle needs to be authenticated before retrieving data from other vehicles.
Integration	High integration. This function is easy to be incorporated into the broader Apollo's application context. We can either use v2x's vehicle-to-vehicle cooperative system (it is also a Client/Server architectural style) to exchange data or add new modules to manage family sharing systems and authorization.	Low integration. In order to embed the new feature, each vehicle needs to add new memory hardware storage for Data storage and Registry submodules.

4.1.3 Implementation Pros and Cons

Implementation I: This implementation follows the Client/Server architectural style as it uses Apollo's centralized cloud server, which connects to the other vehicles in the *Family Share System*. Therefore, we can see the first advantage of this implementation: high scalability. The system has eased handling heavy request loads, extensive connections, data size, and deployments. Every vehicle can send the request to the central server instead of sending multiple requests to every other vehicle in the channel. The Cloud Server makes it possible for ample data

storage. Hence, the system has high scalability when the data size is big since they do not need to store the data of itself and other vehicles. As for the deployment, every time a new vehicle wants to join the channel, it can only send a request to the cloud server instead of trying to connect to every other vehicle in the channel. The second advantage is that it is easy to access the data and request a backup to the cloud server as there is centralized data storage. Although there are benefits to centralized data storage, the downside is that there are some privacy issues. If the cloud server is attacked, all the data stored will be disclosed. Also, the system depends on the network's performance. If the connection between the vehicle and the cloud server is down, the server cannot retrieve data from the vehicle. Lastly, the system depends heavily on the performance of the data management system. If the server does not have an exemplary arrangement of channels and vehicle data, it will encounter corruption and erroneous data will be extracted.

Implementation II: This implementation follows a Peer-to-Peer architectural style as there is no central cloud server, and every vehicle has direct access to the other vehicles in the family channel. There is no single point of failure as there is no central server and data can still be transferred as long as there is still a link between vehicles. As the system does not become highly dependent on the network, it can be primarily resilient against network issues or failing peers by establishing new links to other vehicles. On the other hand, as there is no central data storage, each vehicle needs to store the data internally and may not be storage space-efficient. This implementation has low scalability of request load, connections, data size, and deployments. When a vehicle wants to join the channel, it needs to send requests to every other vehicle in the channel and connect to them. Vehicles have no guaranteed response as the vehicle's connections could be down, delayed, or hostile.

By looking through the details of the two implementation methods and comparing their pros and cons, we know that usability is very important and it is easier for Apollo to implement the enhancement using implementation I as Apollo already has a V2X system and other Cloud Service platform. Therefore, we conclude that implementation I has the best performance overall.

4.2 Stakeholder Analysis

Stakeholder	**Implementation I**	**Implementation II**
Users	Users will be interested in the functionality and security of the centralized cloud server of the implementation. The system integrity must not be comprised and information should not be given access to by unauthorized	Users will be interested in the safety of their information and will be more concerned with sensitive information being obtained by other people. Identification and permissions will be the key factors users

	individuals.	will be interested in.
Developers (Apollo)	Developers will be interested in the implementation of the client/server style and the maintainability and testing of the system. Server uptime and the development of the system may be modified to fit the new architectural style.	Developers will be interested in the deployment of the peer-to-peer style and maintaining the proper security measures to ensure private information is not disclosed to unauthorized users.
Investors	Investors are similar to users and will be interested in the security and privacy parts of the enhancement. If the software is not implemented properly and has security leaks, it would affect the company and decrease its value.	Investors will be interested in the integration and the performance of this feature. Security is less of a concern with the importance of data transmitted being high. The integration of the system, as well as the quality of performance, is a bigger concern.

4.3 Concurrency

The Sharer module has been added to the system as a new module, and additional dependencies are required to accommodate the new module. It adds new dependencies on Routing, Monitor, and DreamView. Sharer will subscribe to Localization, Planning, and CANBus which gets necessary information containing the current position of the vehicle. On the other hand, Sharer gives Routing the shared destination to create a route. For implementation I, the concurrency change lies within the networker and cloud where the channel and data storage is contained in. In implementation II, the data storage is instead stored internally and does not have a centralized server and instead relies on the transmission of data between each vehicle. Besides the additional dependencies mentioned above, the overall system does not have significant changes, and the effect of the enhancement on the concurrency of the existing system is trivial.

4.4 Testing

To ensure the proposed enhancement meets the functional and non-functional requirements, testing is required.
The first level of testing is unit testing[5], which aims to validate that every unit of Apollo, including the added module, performs as designed. In Apollo, a unit may be the children in a subsystem, functions, procedures, etc. The method we perform for the level testing is by using white box testing, which is an analysis of the internal structure of each component[6]. The second level is integration testing, which is a type of testing where software modules are integrated

logically and tested as a group. The grouped modules are mostly the subsystems in Apollo. By conducting the integration testing, we can expose the fault in the interaction between two grouped parts - how well the Family Share is performing within the Sharer subsystem and interacting with the initial subsystems in Apollo. Interactions with other modules should not remove existing files but extend and only create new files in appropriate areas. Testing can be performed to check if other functionalities existing before the enhancement are preserved. We will also use white-box testing for this level. The final level of testing will be system testing[4] where the complete and integrated software is tested. Black-box testing[3] will consider the effects of the enhancement on the holistic view of the Apollo system without peering into the internal components of the system. The enhancement should not affect other modules and their computation speeds and should not introduce a severe computational load that exceeds the capabilities of the current hardware. Testing on data security and privacy should also be performed to ensure that information is secure and cannot be breached by unauthorized individuals. Maintenance will also be tested to ensure that maintenance can be performed with ease and is not too complicated to fix errors or bugs after deployment.

5.0 Effects on Conceptual Architecture

5.1 Impact on High-level Architecture

In order to add a family share channel system on Apollo, building on a new component “Sharer” and subscribing/publishing it to a Cloud Server Channel component, responsible for updating car status (position, velocity, etc.) to the channel and updating destination location is a great approach of implementation.[2]

Sharer → Localization: Localization is the module that provides localization services. It is responsible for locating the vehicle position using the RTK localization method. Sharer is subscribed to Localization so that the channel will always have a clear idea of where the vehicle is at.
Sharer → CAN Bus: CAN Bus outputs chassis status of the vehicle like velocity, rotation angle, heading, etc. Sharer is subscribed to CAN Bus in order to gain information about the vehicle during driving so that when doing the sharing destination, the channel can estimate each vehicle's arrival time to the destination and display it to the users to view.
Routing → Sharer: Routing is one of the core components of Apollo. It is responsible for planning all the possible routes from the current position of the vehicle to the destination and fetching it to Planning. Routing is subscribed to Sharer so that when the user inputs a shared destination to the channel, Routing will receive that input and update the destination to produce new possible routes for Planning.
Monitor → Sharer: The role of Monitor is to supervise the hardware and software components to ensure that all the modules are working without any issues. Thus, the monitor will receive data from Sharer and make sure that Sharer is running healthy and the connection to the channel is secure and well connected.

DreamView → Sharer: DreamView is the human-machine interaction component that connects users and the system. It provides a GUI for the users to view the status of the system and inputs for functions. DreamView subscribed to Sharer to receive information about other vehicles which also subscribed to the channel and display their positions and status on the GUI.

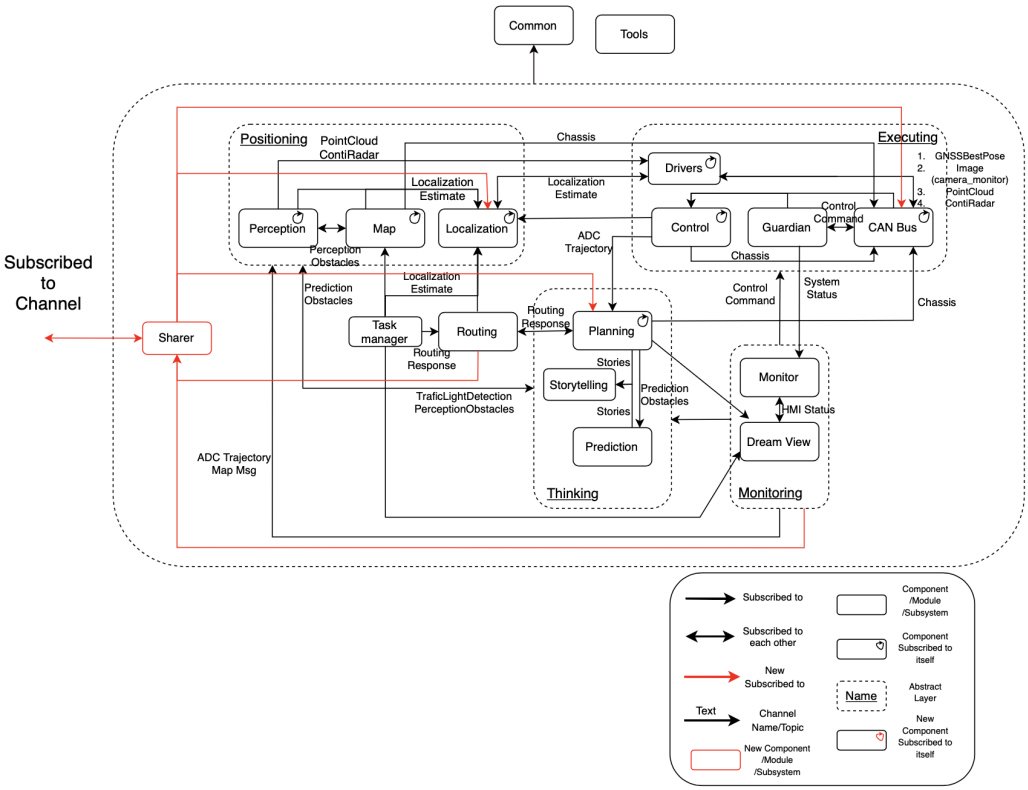

Figure 5. Revised Conceptual Architecture (include *Family Share*)

5.2 Impact on Low-level Architecture

To let the whole system interaction work generally with the new feature enhancement, we believe there are five submodules inside Sharer that will connect to other modules. They are Subscriber, Publisher, Processor, Authorizator, and Networker.

Subscriber → (apollo/modules/planning/planning_component.cc/PlanningComponent::Proc(), apollo/modules/localization/rtk/rtk_localization_component.cc/RTKLocalizationComponent::Proc(),apollo/modules/canbus/canbus_component.cc/CanbusComponent::Proc()): Subscriber will receive the ADC trajectory, LocalizationEstimate, and Chassis Information once those modules publish them.
Publisher ← (apollo/modules/routing/routing_component.cc/RoutingComponet::Proc(<RoutingRequest>), apollo/modules/monitor/software/module_monitor.cc/ModuleMonitor::ModuleMonitor(),apollo/modules/dreamview/backend): Publisher will publish different information for the system, it will update the routing request if some family member decides to share destination in the channel, it will send a message to Monitor about its health status, and it will send data received from channel to DreamView Backend to process and show other family members' location and other information.

Subscriber ⇔ ***Processor***: The subscriber will send the Processor all the information it received from Planning, Localization, and CAN Bus to process or group data like estimated arrival time to the channel.
Publisher ⇔ ***Processor***: Publisher receives data from Processor about other family members' cars and delivers it to Routing, Monitor, or DreamView based on the information.
Authorizator ⇔ ***Processor***: Authorizator is where the system's security occurs where users can decide how much information they would like to "share" to the channel.
Networker ⇔ ***Authorizator***: Networker acts as a port for the Authorizator to authenticate the car user identity so that he can connect to the Cloud Server and channel successfully.
Channel ⇔ ***Networker***: Networker stores the Cloud Server's IP address and acts as a client that requests a visit to the Server so that the system can connect to the channel inside the Server, publish the car information, and receive other family members' car data.

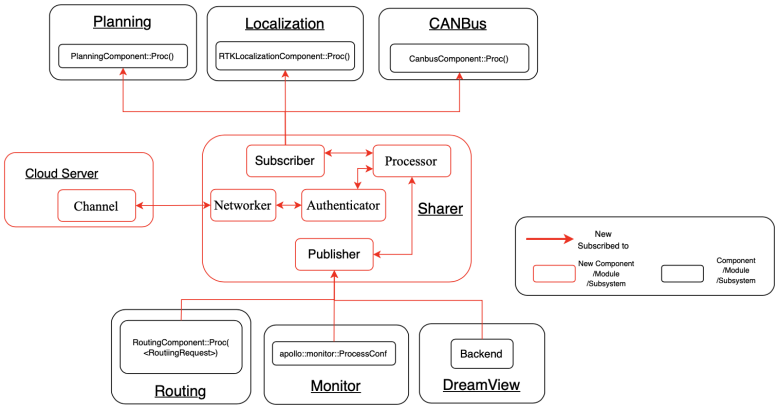

Figure 6. Low-Level Conceptual Architecture SubModule Interaction (With Family Share)

6.0 Use Cases and Sequence Diagrams

6.1 Use case 1: Share Location with Family Members

The user/vehicle attempts to join the family channel, gets the current location information of the other vehicles from the channel, and shares the current location information with the other vehicles in the same channel.
One of the features of our Family Share System is that after the user has joined the family channel, he or she can choose to share the real-time location of the vehicle with the others in the same channel. Meanwhile, the user can also view the current position of the other vehicles willing to share in the same channel. For example, when a group of people is driving multiple vehicles to another city, one followed by another. If a vehicle loses sight of the vehicle in front, the driver can use this function to view the vehicle's real-time location in front to find it. This implementation will allow people to follow each other when they lose sight. We propose this function that can benefit a particular group of people who want to interact with each other.

For this use case, the user (vehicle) first attempts to join the channel, and the Networker submodule in the Sharer module will connect to the Cloud Server and identify the vehicle identity for the channel. If the authentication passes, the user will subscribe to the Family channel and can select what data to be sent to the channel through Authorizator which in this case is the vehicle position, the Localization module gives the vehicle's current position to the Sharer module. Then the Sharer module will give the current position information to HMI to display on the screen. Besides, the Subscriber submodule in the Sharer module will regularly fetch data of other vehicles to get the position information from the channel. CAN Bus would keep track of the vehicle's status. Finally, the information gathered from other vehicles through the family channel will be sent to the HMI for visualization.

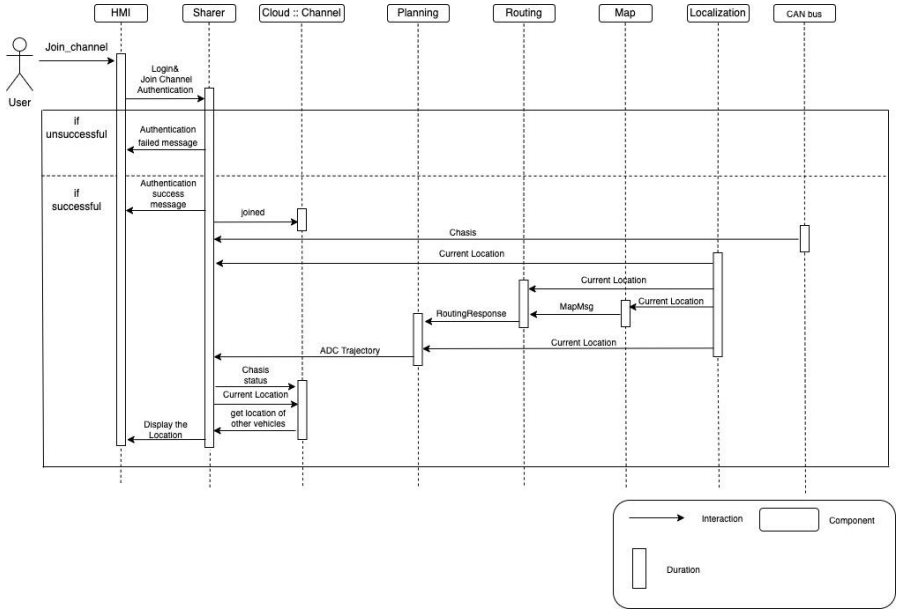

Figure 7. Use Case 1 (Share Location with Family Members)

6.2 Use case 2: Share Destination with Family Members

The user (vehicle) attempts to join the family channel, obtains a destination from the channel, creates a new route, computes the estimated arrival time, transmits data to the channel, and gets an information array from other vehicles.

This use case shows that when family members choose the same destination, the system automatically updates the end location. Based on the current location calculated by the Localization module, Routing will compute new paths and send them to the Planning module, where new trajectories are generated. We also need CAN Bus to keep track of the vehicle's status. Our new module Sharer is responsible for three functions in this use case. Firstly, Sharer gets the desired destination from the cloud. Secondly, the Processor submodule in Sharer will compute the estimated arrival time of the current vehicle to the pre-set destination. Finally, it will gather information from other vehicles through the family channel and send them to Dream View

(HMI) for visualization. In the meantime, Sharer will send information of the user (vehicle), for example, location, trajectory, and estimated arrival time to the cloud for other vehicles in the same family channel to extract.

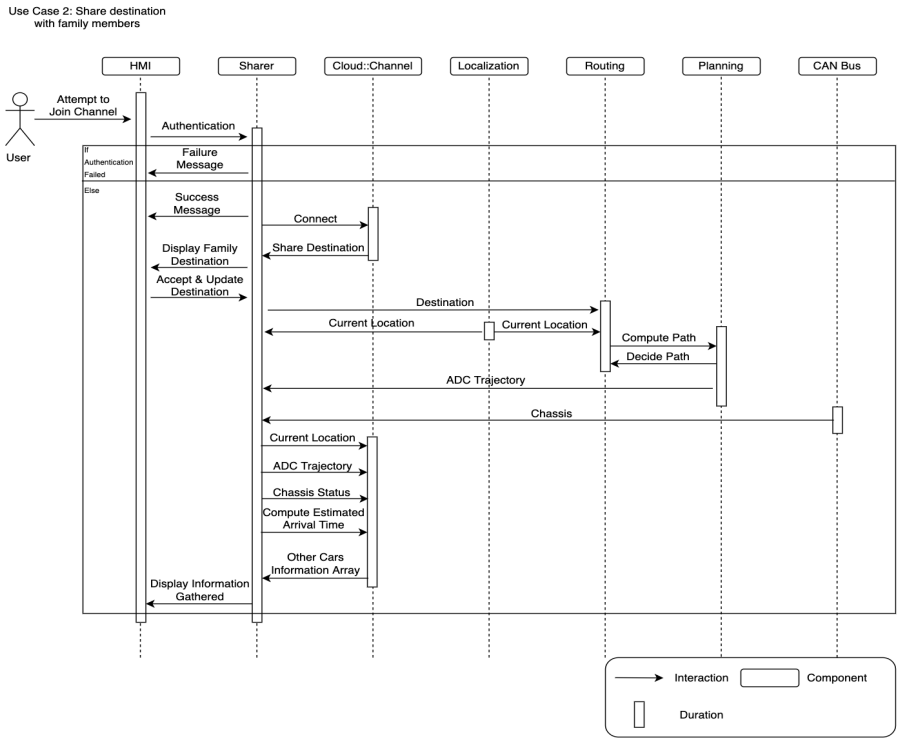

Figure 8. Use Case 2 (Share Destination with Family Members)

7.0 Potential Risks and Limitations

According to the combination of Pub-Sub and Client/Server architectural styles, our new *Family Share System* may face several potential risks and Limitations:

- There is no guaranteed response or idea of what other components will respond to it. It may be caused by the internet being down or other situations. Therefore, the location share function may not work without a stable internet connection.
- Since users may share their locations with other members in the same family channel via the internet, it may cause privacy leakages and are vulnerable to cyber-attacks.
- *Family Share System* is a new subsystem based on the Apollo system. It requires users to create or subscribe to family channels to build connections with other users. Those with a lower version of Apollo System installed or using other autonomous driving systems may not be able to use these functions.

8.0 Lessons Learned

Due to the cooperation of the whole semester, all the members are working more professionally like an elite team. We have given several ideas for our new features by brainstorming, and we

can reach an agreement very soon. The SAAM Analysis requires us to have thorough thinking of our new feature. At first, we still misunderstood our system since we did not ensure the functions in our *Family Share System*. That led to the repeated modifications of the sequence diagrams and use cases. Then, we dug into our new features in every aspect by holding meetings and exchanging ideas and finally nailed down the functions. The experience we gained from this report is something more than having critical thinking about an existing project. We can make improvements and even enrich the system by group working.

9.0 Conclusion

The analysis and benefits of the proposed enhancement have been performed to properly document and evaluate its performance. Functional requirements, non-functional requirements, and stakeholders were considered when performing the SAAM analysis on the enhancement. Additionally, two use cases and sequence diagrams were brought up further to study the potential problems and execution of the enhancement. Finally, by introducing the additional module Sharer, we will be able to put any additional submodules into the Sharer to perform the required tasks to implement our proposed enhancement Family Share.

10.0 Data Dictionary

v2x: Vehicle-to-everything is communication between a vehicle and any entity that may affect, or may be affected by, the vehicle.

Client/server: an architectural style in which the server hosts, delivers and manages most of the resources and services to be consumed by the client.

Subscriber: a submodule of Sharer which subscribes to the channel and regularly fetches data of other vehicles from the channel.

Publisher: a submodule of Sharer which publishes vehicle data to the channel.

Authorizator: a submodule of Sharer which allows the user to choose what data to publish.

Networker: a submodule of Sharer which remembers and connects to the Cloud Server.

Processor: a submodule of Sharer which computes Estimated arrival time.

References

[1] Apollo Auto. (n.d.). *ApolloAuto/apollo: An open autonomous driving platform*. GitHub. Retrieved April 8, 2022, from https://github.com/ApolloAuto/apollo

[2] Kazman, R., Webb, M., & Gregory Abowd, L. B. (n.d.). SAAM:. *A Method for Analyzing the Properties of Software Architectures*.

[3] *Software Testing | Black Box Testing - javatpoint*. (n.d.). Javatpoint. Retrieved April 11, 2022, from https://www.javatpoint.com/black-box-testing

[4] *System Testing*. (2020, September 21). SOFTWARE TESTING Fundamentals. Retrieved April 11, 2022, from https://softwaretestingfundamentals.com/system-testing/

[5] *Unit Testing*. (2020, September 13). SOFTWARE TESTING Fundamentals. Retrieved April 11, 2022, from https://softwaretestingfundamentals.com/unit-testing/#Analogy

[6] *White-box testing*. (n.d.). Wikipedia. Retrieved April 11, 2022, from https://en.wikipedia.org/wiki/White-box_testing