Enhancement Proposal of Apollo

CISC 322 Project Deliverable A3

Date: Apr, 11, 2022

Baik, Daniel

Chen, Oscar

Ma, Yanzhang

Pitcher, Tyler

Rutledge, Josh

Abstract

In this report, we will propose an enhancement to Apollo. The proposed feature is a vehicle-hailing service. This feature will allow the user to call a vehicle to his current location using his cellphone. Firstly, the new feature will be explained more in detail and how it is implemented in the system as well as its benefits. Then, a SAAM analysis is conducted to evaluate the impact caused by the implementations. The result of the analysis will assist in deciding an implementation. Finally, with the help of the analysis, the best way to enhance Apollo can be decided.

Introduction

Combined in this report is the proposal, implementations, and analysis needed to introduce a fully autonomous ride-hailing feature into Apollo. Apollo is an open-source autonomous driving system built collaboratively and shared online for everyone to use. The system is capable of driving to and from locations without human interference. There are even specific subsystems embedded to protect passengers in the case of an accident or to prevent severe road accidents from happening to begin with. The platform promises to provide a flexible architecture able to develop, test, and deploy autonomous vehicles worldwide with ease. The new ride-hailing service in this report will allow users to request a vehicle by providing their current location and desired drop-off location to the system. An autonomous vehicle will then be sent to pick them up and deliver them to their final location. The Silver Doves present two different ways this new ability can be implemented into Apollo's architecture. Both implementations will have an accompanying SEI SAAM architectural analysis in order to determine which implementation is overall the best way to accomplish a ride-hailing service in Apollo. This analysis will include significant stakeholders of the new enhancement to Apollo and the crucial non-functional requirements needed by the stakeholders; furthermore, how each of the two implementations can impact the stakeholder's non-functional requirements will be included in the analysis to understand which implementation should be chosen. The first implementation provides a minimum viable product for customers to be able to request rides from Apollo. It works by altering Apollo's routing module in order to receive routing requests over the air from a new online platform. Users will use the online platform to hail new rides. In the second implementation, we rely on other companies to handle the hailing and scheduling of vehicles. Instead, we focus on altering the monitor module to receive routing requests from online services as user input. Our group knows how an autonomous taxi system will allow non-enthusiasts to experience the awe of driverless vehicles and the benefits of such services to society. We believe Apollo is uniquely positioned to pivot towards providing driverless ride-hailing services.

Taxi Service

To improve the Apollo system and create an enhanced product we decided a taxi service would add tremendous value. Similar to a typical taxi service, customers will request a ride, and an autonomous car using the Apollo system will arrive at their location and take them to the requested destination. Rideshare apps are currently very popular, apps such as Uber and Lyft have taken over the rideshare market and there are likely very few people who would rather call a taxi than use either of these services currently. Autonomous vehicles seem to be the next logical step in further improving the taxi experience. A more consistent

user experience, eliminating the possibility of human error while driving, and creating a more comfortable environment are all possible with autonomous vehicles.

Currently, Uber and Lyft vehicles are driven by everyday people trying to make money. These drivers want to create as much income as possible thus the majority of them go to busy areas. This makes sense as this is where you will get the most rides. The more rides you can complete back-to-back, the more money you will be making. With an autonomous vehicle, there is no desire to complete more rides than other autonomous vehicles. Thus, you can place as many vehicles in a region as you see fit. You can leave a few cars in quieter areas with no complaints.

Entering a stranger's vehicle even just for a short ride may not always be enjoyable. People value their privacy and thus may prefer to be alone. An autonomous vehicle makes for a contact-free environment which can be much more comfortable for customers.

Having a human driver and little requirement to get said position leaves for the possibility of an under qualified driver. There is always the possibility of human error with a driver. This is not possible with an autonomous driver and the experience will always be the same. You are guaranteed a certain level of comfort and safety.

The above demonstrates how an autonomous vehicle could improve rideshare services and thus would be a great addition to the Apollo system. Providing the system with a real-world use that not only creates a more accessible need for Apollo but also improves an already popular industry.

Implementation 1

Apollo's autonomous driving system is in a solid position to pivot toward providing individuals with a fully autonomous taxi service that can be hailed through their cellphone. This proposed implementation will provide a minimum viable product for users to test-drive. Alterations to the system will be needed further to ensure the entire system offers a service fulfilling customer desires. The whole system will operate utilizing two critical systems. Apollo's cars will be the first essential part of this system; furthermore, Apollo will have to build a new online service system. Most of the changes needed to implement this proposed system will occur on the online service front since the car only has to drive to the pickup and drop-off locations.

Introducing a new online platform for customers to hail available cars will provide an essential feature required for this proposed autonomous taxi service to operate successfully. Users will provide their current location to the service. Once a user has entered their area and desired drop-off point, the online service will communicate over the air with the nearest Apollo capable car. The nearest car will need to drive to the pickup location and await further instructions from the online platform. Once a user has gotten into a car, they will notify the online system, and the vehicle will then be notified of the drop-off location. Upon arrival, the user will again inform the online service that they have exited the car, and the car will be notified; furthermore, the car will be told what to do next. These instructions may include where to pick up the next customer or where to wait until the car is needed again.

Minor modifications to the entire autonomous driving system are needed in order to implement this new self-driving taxi system. Since most of the work will be done through Apollo's cloud services, only the routing module requires minor alterations. This is because Apollo is already capable of autonomous driving and would only need help with the ride-hailing portion of the new taxi system. Once the car knows the final destination, it can handle the rest. When a new user requests a ride, the online service will send the nearest car's routing module the pickup location over the air. Then the car's system will handle the rest by driving to the correct area and picking them up; furthermore, once the user has confirmed that they have been picked up by the car, the online platform will then send the car to the user's drop-off spot. Again, once a car has received the desired location, the car is able to drive itself to the right spot, which means the car will be able to drive to the user's destination. As noted, this implementation requires little change to Apollo's car architecture, which in turn means most of the hailing and scheduling services will be done through the online service. With little work needed to be done on the car end already, existing vehicles can be easily updated to participate in the autonomous ride-hailing taxi service.

Due to this implementation having few changes occurring on the car end, there should not be any significant risks; however, no system is perfectly risk-free. There are two potential risks involved in this implementation. The first risk is the introduction of a new threat vector for malicious interference in the riding-hailing service. Relying heavily on instructions from the online service means the car is vulnerable to receiving invalid requests from malevolent actors. An example of this would be overriding the user's chosen destination with another more dangerous location. This vulnerability is a common issue faced by other ride-hailing services. It can be handled by utilizing authentication and encryption to ensure only valid requests to the car's routing module are made. While requiring few modifications to the car's system is beneficial, it is incurred with potential downsides. One of these downsides includes the car driving away without the user riding in the car. This implementation only alters the routing module to receive requests over the air, which means it does not check to see if the customer has gotten in or out of the car. Instead, the entire system requires the user to notify the online platform that they have entered or exited the vehicle. Modifications to the car's guardian or control module can be made to ensure the car does not drive to the drop-off location without the passenger being in the vehicle. Creating a simplified implementation to test-drive an idea comes at the cost of building a non-perfect solution.

Providing an autonomous taxi service through Apollo's already existing architecture can be accomplished while creating the minimal amount of complications. The service will require an expansion of Apollo's online cloud platform to allow for hailing and scheduling of autonomous taxi cabs; furthermore, a minor alteration to Apollo's routing module is required in order to receive routing requests over the air from the online platform. This implementation of an autonomous riding hailing service will introduce few risks that do not already exist in the system.

Implementation 2

In the first implementation method, we provide the technical implementation method, and we will also provide a further method in the real world.

Developing a fully autonomous taxi service needs more resources and more testing time. Although the service is completely developed, due to this taxi service is meant for the public rather than the vehicle buyer, it needs to compete the same function application, so the publicizing fund such as advertising fee, discount to the passenger and so on. According to the news about the competition between DIDI and Uber in China in 2016, the cost of the publicizing fund is more than 4 billion US dollar per year to DIDI and Uber even lost 1 billion US dollar in 2016(https://www.sohu.com/a/108616804_114837). The cost of the publicizing is much more than the development resource we have and is a huge pressure to Baidu. What's more, in order to maintain the taxi service, a service group must be form, and it's include a large amount of 24-h customer services, an 24-h technical maintenance team, a 24-h emergency response team which used to solve the traffic accident, and in-person customer requirement in each city which the taxi service cover, and many administration staff to run this service group. It will also rapidly increase the cost.

In order to reduce the resource input on this area, the best way is to set up a communication portal with other platforms which provide the similar service, or import a complete application which is designed by other companies, which own similar services. This communication is similar to the cooperation between Google map and Uber, so we assume the communication on the Apollo system is between Baidu map and DIDI, and it has two ways to set up this communication.

In the first way, we set up a communication portal in the systems. Because the taxi service is not related to most of the modules in the Apollo's architecture except the Map Engine module and Planning module, because the taxi service platform is provided by the other company, they only need to know the start position and end position. So, the communication portal can be set as a new module in the architecture which depends on the Map Engine module and Planning module. The Map Engine provides the location and the Planning module receives the outer information to set up the plan. On the other hand, in this situation, due to the Planning module needing to have more dependence to other modules and its the core module to whole autonomous systems, it will form some risk such as bug and Trojan horse if the outer module directly connects to these modules. The safest way is to create a new firewall module between these two modules and portal module, but it will increase the complexity in the systems.

In the second way, we import an application module to the monitor module, this module acts as the driver's input. Once the back-end of the application receives the information created by the customer, it will autonomously provide the start position and end position to the Apollo systems, and Apollo systems will complete the rest requirement. In this situation, this module is only related to the Monitor module, so it reduces one dependence between the new module and the whole architecture, so it can reduce the complexity compared to the situation above. This way is also have same risk problem as the first way and due to the monitor module is related more module than map engine and planning module, the firewall module between monitor module and new module is necessary and need to be stronger.

SAAM Analysis

The SEI SAAM architectural analysis, Software Architecture Analysis Method, is a method for analyzing architectures that evaluate and assess the general architecture. The paper describes three software architecture perspectives and proposes a five-step process for analyzing software architectures. Using SAAM provides the opportunity to find problems in the architecture and help preserve the qualities and non-functional requirements of the stakeholders.

The five activities involved in the SAAM are:

1. Characterize a canonical functional partitioning for the domain.
2. Map the functional partitioning onto the architecture's structural decomposition.
3. Choose a set of quality attributes to assess the architecture.
4. Choose a set of concrete tasks which test the desired quality attributes.
5. Evaluate the degree to which each architecture supports each task.

We choose four key performance indicators for the proposed feature: Maintainability, Compatibility, Testability, and Security. And our SAAM includes several steps: first, selecting a set of non-functional requirements that we can use to add enhancement features based on those requirements. Second, we compare the two implementations to see which is promising. Third, propose test cases to show the impact on the architecture of the implementation. At last, we discussed the impact of the architecture and the stakeholders.

NFR	Effect (Implementation 1)	Effect (Implementation 2)
Security	High, adding the taxi module would be under the watch of cloud service. It's a module in the distributed system that spies on each process related to Apollo. In addition, the implementation has few changes occurring on the car end, so there is a low chance of information leaks.	Low. As the taxi driver platform is more like a plugin, we cannot ensure the safety of the third-party developers; using the plugins will cause severe problems with protecting the user's information. Also, the security measures will be controlled by third-party applications.

Compatibility	High, the developers implementing other modules will implement the taxi module. Therefore the compatibility will be high. And developers will update the publish and subscribe technique of specific modules to make the new taxi module work.	It depends, but usually, third-party compatibility can be high because modern taxi platforms like uber have well-functioned APIs that can fit the Apollo. But there will be some issues to fix due to the version updates or further feature enhancement.
Testability	High, the pub/sub architecture allows each module to work as its functions, and there are many dependencies that the developer could test.	Low. The third-party plugin contains various functions, and the source can be used on multiple platforms. So, it isn't easy to test independently.
Maintainability	High. The new feature works as an individual module like the newly introduced storytelling module. The developers only need to maintain the module, and other related dependencies will be enough.	Low. The change of anything in plugins could cause an update for the third-party plugin. If the third-party plugin is updated, the apollo also needs to update itself to match the new API. It can be considered a difficulty for maintenance.

Stakeholder	**NFR in Project (Implementation 1)**	**NFR in Project (Implementation 2)**
Users	Users will focus on the reliability and availability of this implementation. Users with no available vehicles are the main stakeholders of the taxi service. Therefore, an autonomous taxi with good reliability and could handle all situations are the primary concern.	Users will focus on the third-party platform and its quality. Including whether the platform responds quickly, can travel in minimum time, and chooses the nearest taxi is their requirement.

Third-Party Developers	Developers will focus on the complexity of the architecture and its codes; tight and simple architecture causes high stability with low coupling. Apollo open-source platform has excellent adaptability; therefore, additional module implementation can be easily accomplished.	The developers will focus on the maintainability and serviceability of the features in case of further development and enhancement. The corresponding platform may update quickly so developers will need the open-source SDK for additional updates or maintenance.
Apollo	Apollo will focus on its security and regulatory since the platform aims to have a well-organized system. The consistency of R&D, planning, and operations is their primary concern.	Apollo will focus on security and data integrity. They will keep monitoring the third-party plugins to avoid an information leak.

Comparison of 2 implementations

	Implementation 1	**Implementation 2**
Pros	1. It's an Apollo original feature that will attract more customers and users for its convenience. 2. The developers can perform good compatibility with any other functions that Apollo can provide. Also, the powerful cloud service system of Apollo can be handy.	1. The cost of implementation is saved. 2. Users will prefer a well-known platform rather than a new platform. 3. Can reduce the usage of the own cloud service as well as the pressure of the management team.

	3. The security can be guaranteed.	
Cons	1. The service can be extensive, including customer service, technical services, maintenance team, and emergency response group. This implementation will bring a new department for the taxi service, which will cost a lot. 2. The autonomous driving is still in early progress. Therefore, taxi service can be a more significant challenge. 3. More modules for the system might cause maintenance more difficult. And the taxi module will be associated with various modules, which increases the difficulty of implementation.	1. The security may not be guaranteed if the third-party platform handles the entire service. 2. The Apollo platform evolves as well as the third-party platforms. The difference between versions could lead to more work on the plugins. 3. The third-party platform will also bring more complex agreements.

For Apollo, the better solution is still to connect with third-party softwares. This solution not only does it save a lot of money, but also third-party software can better attract customers.

Impact on the Architecture

The addition of this taxi service will improve the Apollo system. With a network of cars driving in similar areas all communicating with each other, you provide far more information to your system thus cars can accurately read traffic and accident information from areas they have not yet traveled to. Improving the Apollo system by creating a smarter network. Allowing for more accurate ride time estimations and better-planned trips.

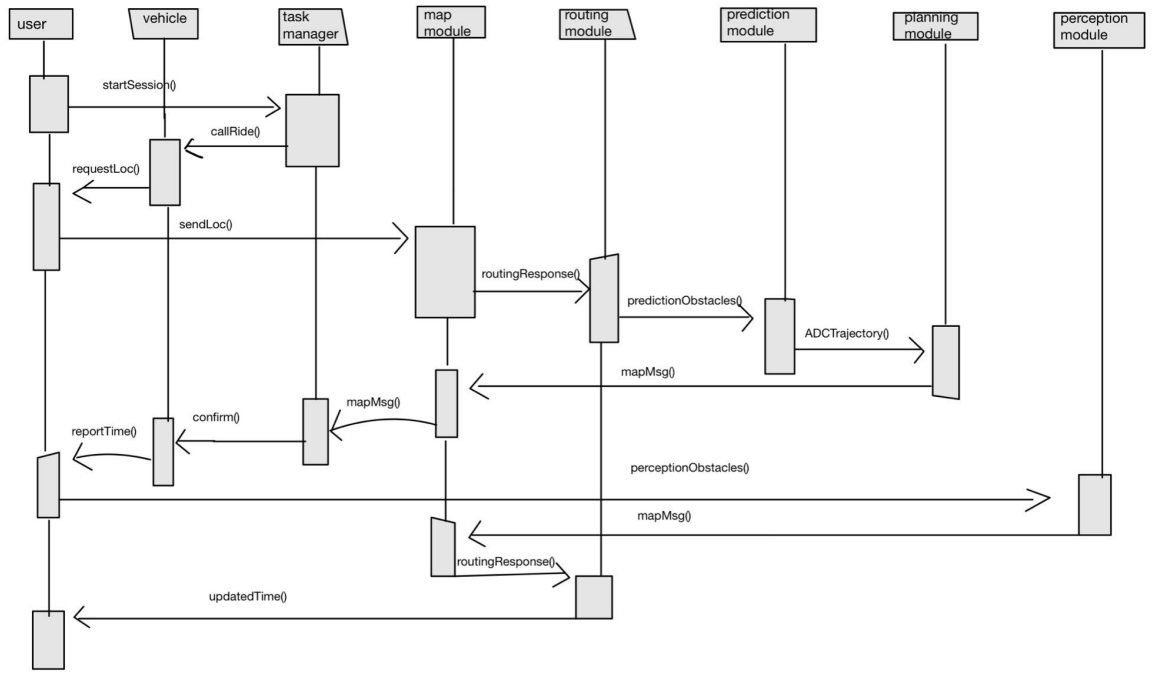
In the current Apollo system cars receive information from on-vehicle sensors. These sensors track immediate occurrences and allow a car to update its planned path in real-time. If a car could read the data of a car on its current path but 10 minutes ahead it could make better predictions and when you have a network of cars, you have even better predictions. For example, a car is close to point A and has a planned route to point B. Consider another car that has traveled from point A and is now close to point B. Car 1 reads its immediate traffic data and everything seems to be moving regularly but Car 2 reads a lot of traffic close

to point B. This is okay for Car 2 as it is so close to its destination and no course readjustment needs to be made but Car 1 can reroute itself to avoid the traffic and arrive at the destination much quicker.

In order for this to succeed the architecture will need to change. It will require all cars to be connected to a common network where they can both receive and upload data. Most importantly a car should understand where it is in regard to other cars and read traffic and accident information from cars ahead of it on its current path and a few alternate paths; to determine the best of multiple routes. Cars must also have 2 states, in-use and idle this way cars can determine where to be to better alleviate traffic and provide customers with good service.

An in-use car will travel to the chosen destination, choosing the best possible route. An idle car (car with no passenger) will still need to be driving to prevent traffic buildup and a lack of parking. These cars should also attempt to place themselves in positions where they prevent traffic for in-use cars and also are highly available to customers. For example, an idle car should never be on a highway as it will not be able to pick up any riders. The car should also not be driving in circles around a popular restaurant waiting for a customer to exit, this would build up traffic.

Use Case



The sequence diagram is for a use case of a user calling for a vehicle to his location using his cellphone. First, the user calls for a vehicle by starting a session and using the callRide function which will let the vehicle know of the user's call. Vehicle then requests the user's location (requestLoc), getting the GPS coordinates from the user's phone. The location is

sent to the map module (sendLoc), then to the routing module (routingResponse) and at this point, a route to the user has been created. Next, the route is more precisely planned out using the prediction module (predictionObstacles) and the planning module (ADCTragectory). The precise plan is sent back to the map module (mapMsg) and eventually reports estimated arrival time of the vehicle to the user (reportTime). Because new obstacles may incur while the vehicle travels, the perception module is utilized again, reporting the changes to the map module. The finalized route is created from the routing module and new arrival time is updated for the user (updatedArrival).

Conclusion

For the new function taxi system, we propose two methods, one is to add a taxi module in apollo, and the other is to add a third-party taxi system. Comparing the advantages and disadvantages of the two methods, it is not difficult to find that the taxi module in the process not only has high compatibility but also has high testability. Compared with adding a taxi module to the software, it is more convenient to connect with the third-party taxi system, which can save a lot of manpower and money and attract customers. The fly in the ointment is a potential security problem. In general, a mature taxi system can make autonomous driving more user-friendly and convenient.