Concrete Architecture Report on Apollo

Minxuan Li 20144519 18ml51@queensu.ca
Li Bowen 20144417 18bl29@queensu.ca
Steven Wen 20144322 18yw85@queensu.ca
Yuehan Qi 20167259 18yq36@queensu.ca
Aolin Zhou 20058020 16az31@queensu.ca
Date 2022.3.21
Queen’s University
CISC322
Instructor: Prof. Bram Adams

Table of Content

1. Abstract
2. Introduction and Overview
3. Derivation Process
4. Top level Concrete Architecture
    - 4.1. Recap of Conceptual Architecture
    - 4.2. Concrete Architecture
    - 4.3. Modified Publish-Subscribe Architecture
    - 4.4. Reflexion Analysis on Top level Architecture
5. Subsystem Analysis: Monitor
    - 5.1. Inner Architecture Analysis
    - 5.2. Reflexion Analysis
6. Second Level subsystem Analysis
    - 6.1. Interactions of Second Level Subsystem On-Lane Planning
    - 6.2. Rationale
7. Sequence Diagram
8. Alternative Architecture
9. Naming Conventions
10. Conclusion
11. Lessons learned and Team Issue
12. References

1. Abstract

For the previous paper, the Baidu Apollo Auto driving technology platform was analyzed for its conceptual architecture, including functionality, software system, data control, and system concurrency. Deriving the analyzed conceptual architecture and using Understand to analyze the source code, this technique report provides an extent to the concrete architecture. In this report, three new subsystems are added: Routing, Task Manager, and Storytelling. After appending the new subsystems, new dependencies with other subsystems are discovered. Applying the inner architecture analysis and reflection analysis, the initial proposed conceptual architecture is imperfect, and this report analyzes unexpected dependencies and interactions to improve the previous architecture. Eventually, this report includes the limitations and derivation of how tools are used.

2. Introduction and Overview

Apollo is an open-source platform to help car industries build their autonomous driving system by combining their hardware with “Apollo”. Apollo comes from the “Apollo program,” wishing it could be a milestone for the self-driving industry. Apollo’s environment perception, path planning, vehicle control, and in-vehicle operating system are the core modules that rely on each other. The developer needs to configure all the modules before testing the vehicle. As for now, Apollo has released its 7.0 version, which has three brand new deep learning models to enhance the capabilities for Apollo Perception and Prediction modules which helps it drive on complex urban roads.

This report aims to provide an overview of the concrete architecture of Apollo by analyzing the discrepancy of its previous conceptual architecture using inner architectural analysis and reflexion analysis. Compared with conceptual architecture, the Understand is used to extract the diagram of a concrete architecture that can better understand the architecture. After looking at the diagram, three new subsystems, which are Routing, Task Manager, and Storytelling, are determined to append in the architecture. Since this report changes the subsystems, some unexpected dependencies are found. Due to the unexpected dependencies, the modification of the conceptual architecture is made to propose a better concrete architecture. This report pays much attention to the Monitor subsystem and second level sub-system in the Planning subsystem, through the new views on the concrete architecture since it contains system-level software such as code to check hardware status and monitor system health. Combining the conceptual and concrete view, two use cases are proposed to show how each part interacts with others. Finally, an optimal perspective of concrete architecture is provided.

3. Derivation Process

To better know the concrete architecture of the Baidu Apollo Auto driving technology platform, the SciTools Understand is used to generate the dependency graph. Understand is a code evaluation device that lets the users visually examine the dependencies of a codebase. After knowing this application, the code from Apollo is mapped to this tool and generates a dependency graph to understand each module's interaction better. By analyzing the interaction of each module, unexpected dependencies are found. As we can see from the dependency graph, the interaction is really complicated, which means that it is unrealistic to analyze each dependency carefully. In this case, only significant and essential dependencies are analyzed in our concrete architecture, and the trivial part of dependencies is ignored for the sake of efficiency. After mapping all the code and based on the graph visualization of pub/sub message traffic provided, we came up with a better understanding of the Apollo architecture.

4. Top-level Concrete Architecture

4.1 Recap of Conceptual Architecture

Based on the analysis in the previous paper, the conceptual architecture is Publish-Subscribe Architecture.(Figure 1)

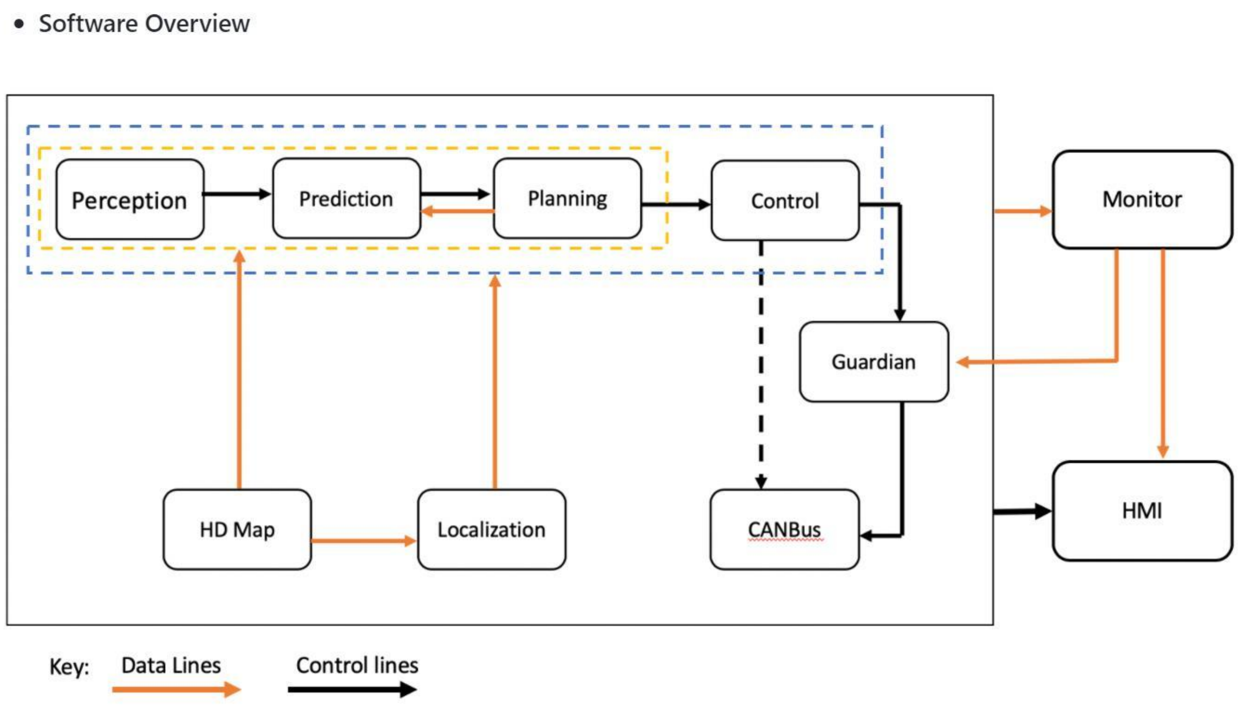


Figure 1. Overview of the Apollo Conceptual Architecture.

4.2 Concrete Architecture

As the provided graph visualization of pub-sub message traffic, the concrete Architecture is illustrated in the figure below.(Figure 2)

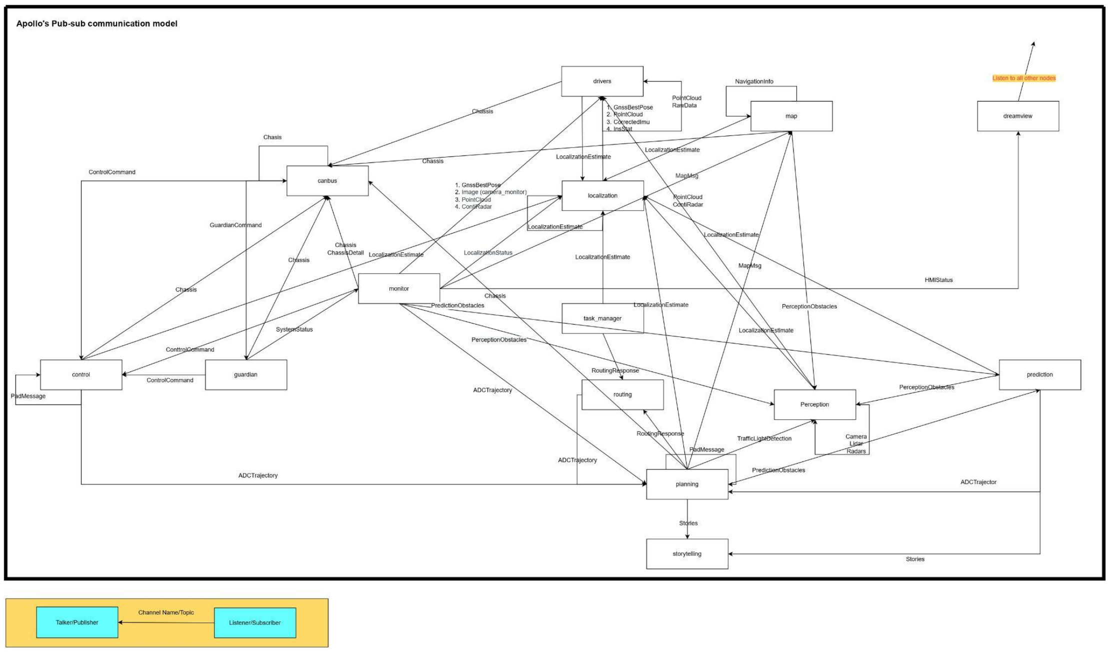


Figure 2. Graph visualization of pub-sub message traffic

4.3 Modified pub-sub Conceptual Architecture of Apollo

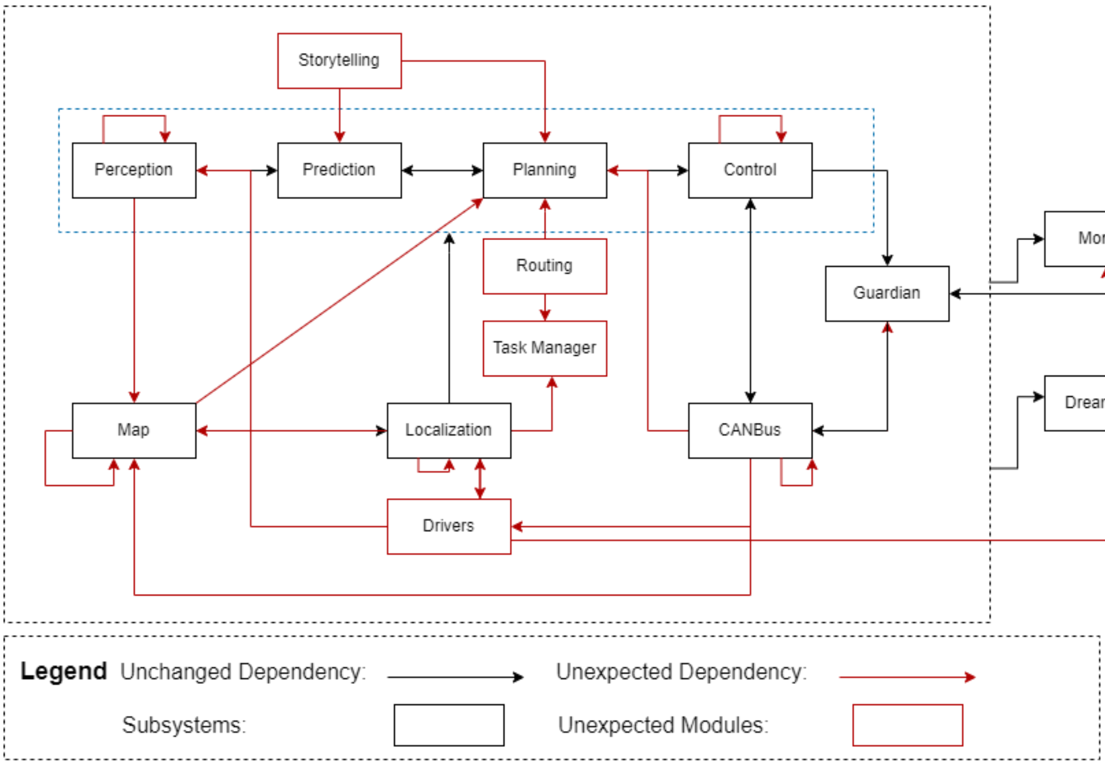


Figure 3. Modified pub-sub Conceptual Architecture of Apollo

The modified conceptual architecture still exhibits a pub-sub style. Three new modules emerged to help specify the connections between the existing modules in the original conceptual architecture. Furthermore, it is noticeable that the driver is not counted in the conceptual architecture.

4.4 Reflexion Analysis on the Top-Level Architecture

4.4.1New Modules

1) **Routing**

The routing module takes in a routing response from the task manager and the planning module. Then in return, it produces an ADCTrajectory back to planning.

2) **Task Manager**

The task manager mainly acts as a module that outputs information to other modules. It connects with the localization module by sending the localization estimate to it. It also sends routing responses to the routing module for it to analyze.

3) **Storytelling**

The storytelling module is a high-level scenario manager that coordinates cross-module actions. It takes in complex scenarios from localization and map, then outputs stories that are subscribed to the planning and prediction modules.

4) **Drivers**

The drivers module represents the commands and feedback coming from the driver, which interacts with 4 modules that either receive information from the driver or transmit data to the driver.

4.4.2 Unexpected Dependencies in each module

1) **Map**

Rather than sending data to perception, prediction and planning, the map module actually only sends map messages for planning. Furthermore, it receives information from more resources than expected. With the inputs of perceived obstacles and localization estimates, the map updates its navigation information.

2) **Localization**

Instead of receiving one-way input from Map, there is a mutual connection between the map and localization modules. The localization module sends the localization estimates to the map and itself for developmental accuracy.

3) **CANBus**

Besides the established relations found in the conceptual architecture, another major function of CANBus is collecting the car's chassis status then reporting the status to the planning, map, control and guardian modules for safety control.

4) **Dreamview**

Dreamview is the HMI module that helps visualize the output from all the other modules. It interacts with all of the modules in Apollo software by monitoring their output messages. However, instead of simply perceiving information from the monitor, it also outputs the HMI status to help build a 3D rendering of the monitored messages.

5) **Others**

Apart from the new modules created in the concrete architecture and several modified connections stated above, the rest of the unexpected dependencies are information that modules send to themselves. For example, control conveys pad messages to itself and perception receives perceptions generated from Camera, Lidar and Radars.

4.4.3 Rationale for the Differences

1) **Interaction between Perception and Map**

First, the "HD Map" in the original conceptual architecture is modified to "Map" which includes the pnc map and other relative maps apart from the HD map. Second, instead of the map sending data to perception, the perception module would report the found perception obstacles to the map to mark down. Then the map will send the updated map information to localization and planning for better accuracy.

2) **Missing the Drivers module**
The driver module was not taken into account in the first proposed conceptual architecture because it was considered as a part of the user interface. However, the drivers module also contributes to receiving chassis status from the canbus, which is essential to the safety concern of the vehicle.

5. Subsystem Analysis: Monitor

5.1 Inner architecture analysis

5.1.1 Goals

The monitor is the most critical subsystem in the Apollo since it contains system-level software such as code to check hardware status and monitor system health. It surveys the working condition of the system, and it receives data from multiple modules and checks if they all work without any issue. Then it passes on the information to HMI for the driver to view.

5.1.2 Modules

The monitor is divided into three main modules:

1) Hardware module: Hardware-related monitoring, e.g., CAN card / GPS status health check. Check results are reported back to HMI.
2) Software module: includes three types of monitors:
    a) Process monitor: checks if a process is running or not.
    b) Topic monitor: checks if a given topic is updated typically.
    c) Summary monitor: summarizes all other specific monitor's results to a simple conclusion such as OK, WARN, ERROR, or FATAL.
3) Common: this includes the monitor manager to the monitors.

5.1.3 Conceptual View:

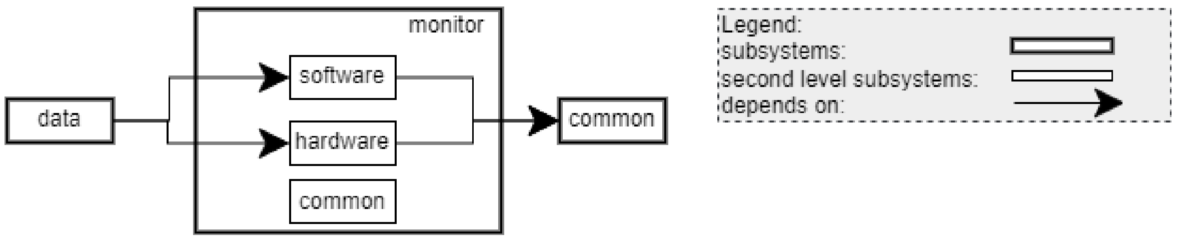


Figure 4. Conceptual Structure of Monitor subsystem

The data is obtained based on the software and hardware monitors of the monitor subsystem and then transmitted to the common subsystem.(Figure 4)

5.1.4 Concrete View:

The figure below(Figure 5) illustrates the monitor subsystem. Through the Understand we extract the dependency structure of it. It has three modules in it, and the data subsystem depends on the common module, while the common module itself depends on the dreamview subsystem.

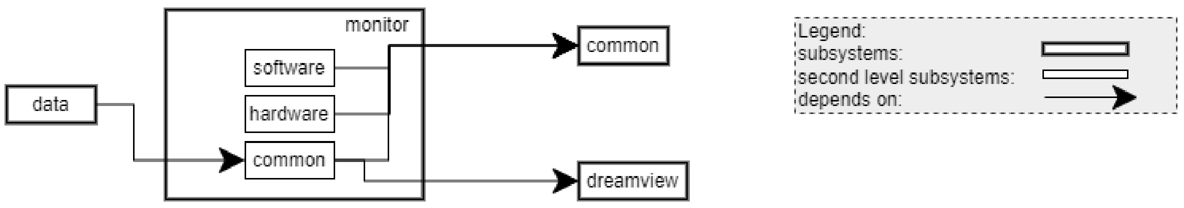


Figure 5. Concrete Structure of Monitor subsystem

5.2 Reflexion Analysis

5.2.1 Divergence

According to the figures above, the divergences are illustrated in the figure below(Figure 6).

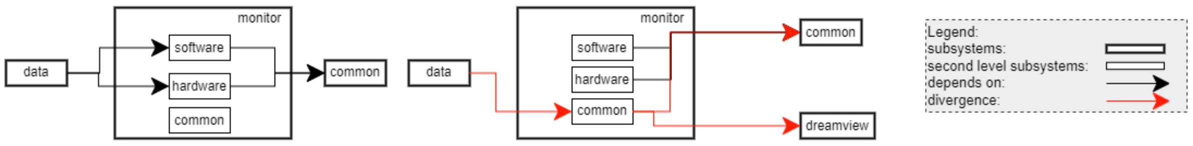


Figure 6. Reflexion model on Monitor subsystem

There are three unexpected relations in figure 6: the dependency from data subsystem to common module, the dependency from common module to common subsystem, and the dependency from common module to dreamview subsystem.

5.2.2 Rationale

As discussed above there are three unexpected dependencies in the monitor subsystem. Next, we are going to analyze the rationale behind these.

1) Rationale for data -> common

In the conceptual architecture, we consider that the data subsystem depends on two modules of the monitoring subsystem, i.e. software and hardware. Instead, it can depend on only one module, the common module. In the previous section we did not explore in depth, what are the specific functions of the common module, which actually contains a monitor manager, after managing the monitors in the software and hardware, the managed streamlined data is used for the data subsystem dependencies.

2) Rationale for common -> common

The common modules within the monitor subsystem need to rely on information related to the operation of the Apollo, and the common subsystem contains code that is just as useful for the operation of Apollo. By looking at the source code, we found that the common modules depend on four specific modules within the common subsystem, the monitoring log (a logging system), the util (contains an implementation of the factory design pattern with registration, some string parsing functions, and some utilities for parsing protocol buffers in files), the configs (specifies the vehicle configuration), and the adapters (different modules use themes to communicate with each other).

3) Rationale for common -> dreamview

The common module in the monitoring subsystem needs information about autonomous driving, for example, dynamic web-based 3D rendering to manage it. Meanwhile, Dreamview, Apollo's human-machine interface module, exactly provides a web application that helps developers visualize the output of other related autonomous driving modules. This causes the unexpected relation in Figure 6.

6. Second Level Subsystem

6.1 Interactions of Second Level Subsystem On-Lane Planning

We chose on-lane planning as our second level sub-system, that's within the planning module. The goal of this second level subsystem is to be responsible for automatic driving along lane lines under open roads in a safe and reliable manner. Three major components are enclosed at this level, including OnLanePlanning, Polygon2D, and Status. Among OnLanePlanning, some sub-components (as illustrated in the following diagrams) are interacting with each other within the second level subsystem, as well as other subsystems in the Planning module. Other than this, these sub-components interact with modules of Canbus, Common, Cyber RT, Dreamview, Hd Map, and Routing to receive valuable information such as vehicle status, vehicle state, trajectory point, etc. The Planning module starts by calling Proc method in the PlanningComponent function to register PlanningComponent to the Cyber Register module, then it subscribes to the prediction obstacles in the prediction module, chassis information in canbus module, and localization estimate in localization module.

After this is done, the Init method in PlanningComponent implements the allocation of concrete planners, Assigns specific planners for the planning module and launches the reference line provider. Upon starting, the reference line provider will start another thread to perform a scheduled task, providing the reference line provider every 50 milliseconds. The Planning module selects different Planning implementation modes according to the configuration. "FLAGS_open_space_planner_switchable" and "FLAGS_use_navigation_mode" are in the conf directory of the Planning module. By default, both configurations are false, therefore the OnLanePlanning implementation is introduced. The most major sub-component in the OnLanePlanning second level subsystem is the plan sub-component, it provides multiple candidate reference lines for Frame object pointers, and it calls the Planner object to get a set of available reference line information. Based on the reference line information, it selects the best reference line and converts it to driving decisions that can be used by the control module.

The second most important sub-component is RunOnce, it receives chassis information such as vehicle speed, vehicle position, localization information, HD Map information, and all the information that is necessary for planning. Each time OnLanePlanning is executed based on the following two inputs, the context information of the planning module, and Init Frame structure (vehicle information, location information, etc). Interactions are taking place between RunOnce and Plan sub-components to allow various information of the vehicle being stored and updated. Init Frame are then initialized by RunOnce; if everything works fine in this subsystem, it calls out the PublishPlanningPb function in the planning module to publish a computed planned path to the control module. As we learned in the lesson, this kind of style corresponds to a Publish-Subscribe style, since any message published to a subscribed topic in the planning module is immediately received by all of the subscribers including the control module in this case.

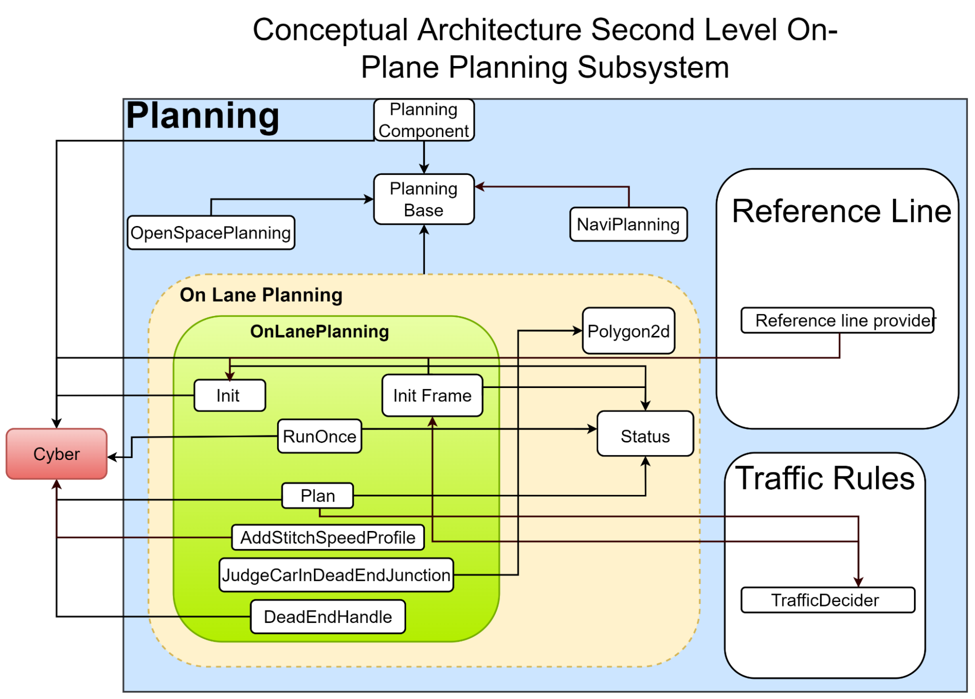


Figure 7. Conceptual Architecture of On-Plane Planning Subsystem

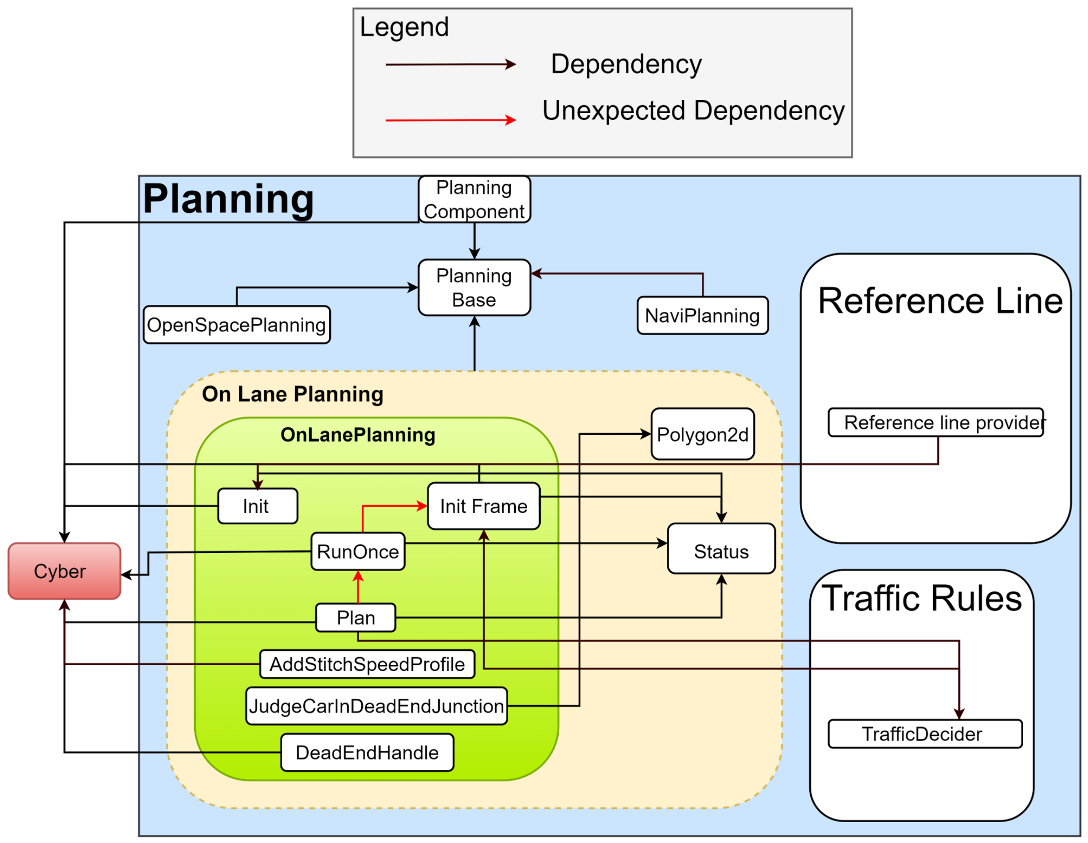
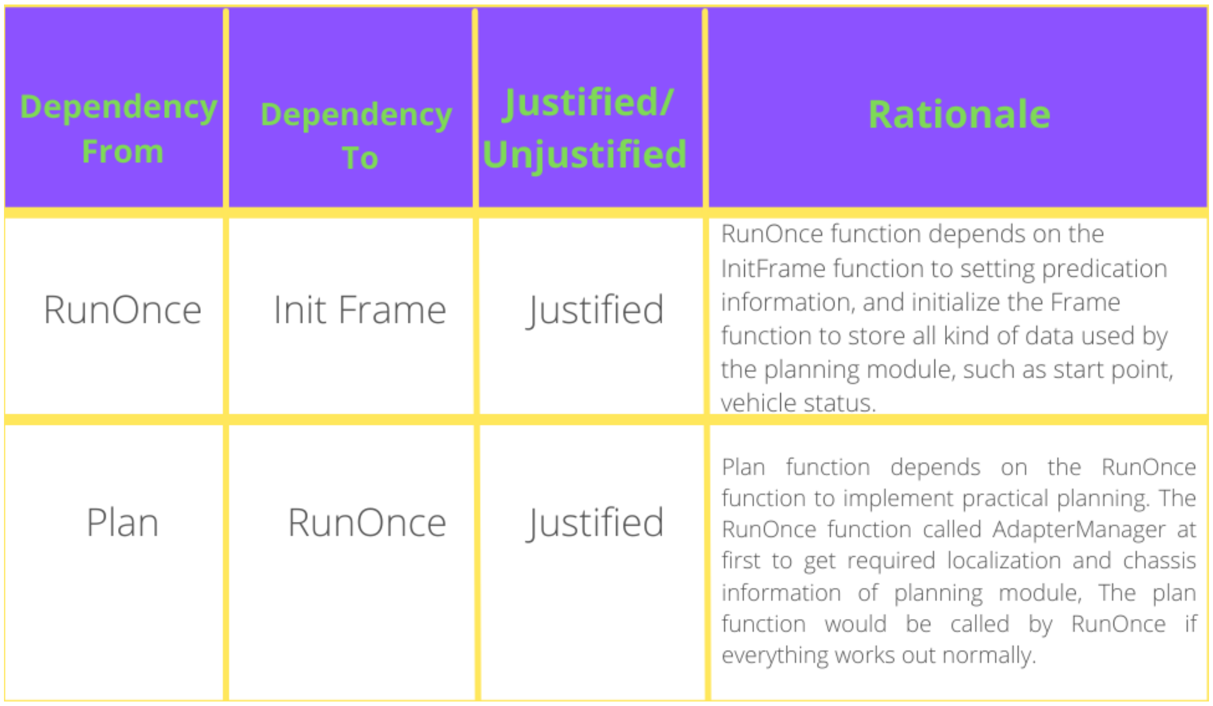


Figure 8. Concrete Architecture of On-Plane Planning Subsystem



Dependency From	Dependency To	Justified/ Unjustified	Rationale
RunOnce	Init Frame	Justified	RunOnce function depends on the InitFrame function to setting predication information, and initialize the Frame function to store all kind of data used by the planning module, such as start point, vehicle status.
Plan	RunOnce	Justified	Plan function depends on the RunOnce function to implement practical planning. The RunOnce function called AdapterManager at first to get required localization and chassis information of planning module, The plan function would be called by RunOnce if everything works out normally.

Figure 9. Second Level Subsystem On-Plane Planing Reflexion Analysis

6.2 Rationale

6.2.1 Rationale 1

We came up with a conceptual architecture for the second level subsystem On-Lane Planning as the figure 7 illustrated. As we dived deeper to explore the code in tools such as Understand, the RunOnce function actually depends on another function called InitFrame to set up all kinds of required information, such as prediction data, start point, vehicle status, reference line information list, and driving reference line information. Upon the necessary data for the planning module have been set up, InitFrame are called by RunOnce to initialize the Frame function to store this data, thus it could be used for the planning module afterwards.

6.2.2 Rationale 2.

A function named AdapterManager is called by the RunOnce function at first, it is used to extract the required localization and chassis information for the plan sub-module. The plan sub-module implements practical planning of the path on the basis of this data. Therefore, we figured out that there is also a dependency between Plan and RunOnce sub-modules that Plan sub-module depends on the information achieved from RunOnce sub-module.

7. Sequence Diagram

Based on the presented concrete architecture, our team has come up with two sequence diagrams. The first one is the general case of Apollo smart transportation, and the second is Apollo’s traffic light perception.

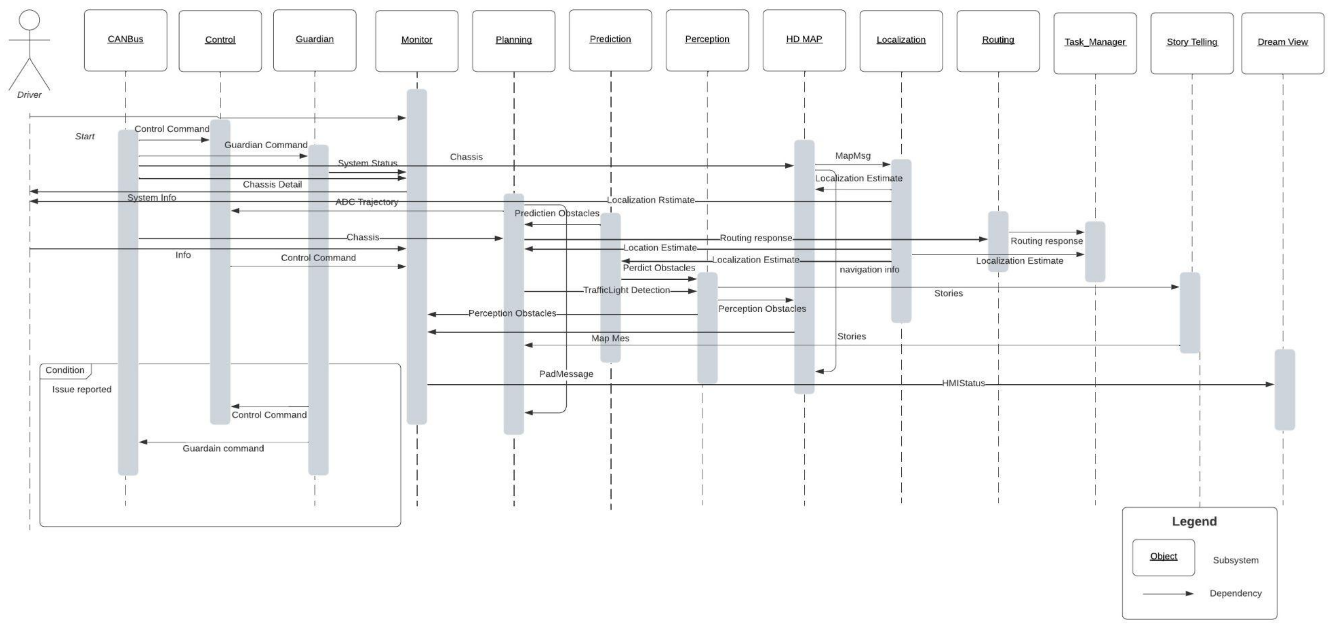


Figure10. The general sequence diagram for apollo

This sequence diagram is similar to the first sequence diagram with minor adjustment. Compared to the sequence diagram from conceptual architecture, routing, task manager, storytelling and dream view has been added. The driver will interact with the monitor with the information then the monitor will subscribe to CANbus, planning, Perception, prediction, localization and map, where the collected information will be sent back to the driver. The perception module will take the information from map, driver, localization and its own camera, lidar and radars data and output the perception obstacles to the prediction module. Where the data will be processed and then sent to the planning module and storytelling. Planning module will be subscribed to CANbus for the chassis information, localization for location estimate, map for map message, perception for traffic light detection, stories from storytelling, routing response from routing and prediction for prediction obstacles. With all the input information planning will output the ADC Trajectory to the control module which has been subscribed by CANBus where the vehicle will be controlled by. Meanwhile guardian will subscribe to the monitor to receive system status if if all the modules are working fine then the Control can be sent to CANBus normally, otherwise Guardian will prevent Control signals from reaching CANBus and bring the car to a stop. The new added module task manager will subscribe to the localization and routing module and manage tasks and make decisions.

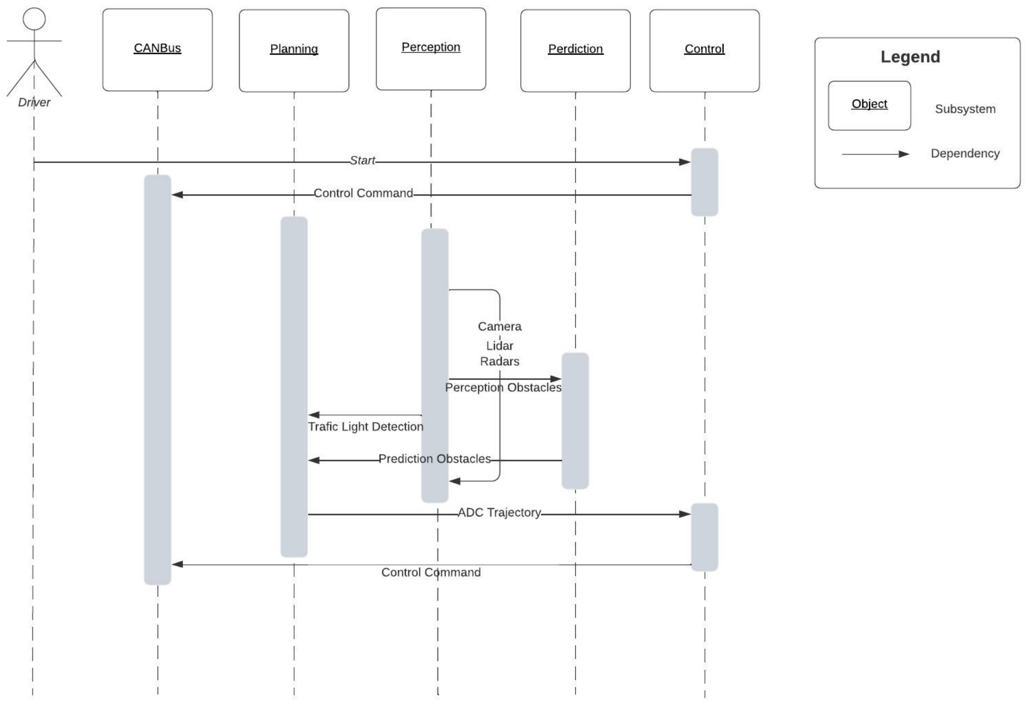


Figure 11. Use case for traffic light

We took a top level view of the traffic lights use case other than the 2nd level we did in assignment 1. This use case will start when the car is in operation, the onboard camera, lidar and radar will be operating consistently. The perception processed traffic light data will be received by the planning module to plan the route for the vehicle, before the planning sends the data to the control module to operate the vehicle the planning module will also receive data from the prediction module where it predicts the safety of the area. After all the planning is done the plan (ADC Trajectory) will be sent to the control module where the control command will be sent to the CANBus to operate the vehicle.

8. Alternative Architecture

As the advantage of Pub-sub architecture is obvious, it provides strong support for reuse since any component can be introduced into a system simply by registering it for the events of that system. Pub-sub Architecture also eases system evolution since other components may replace components without affecting the interfaces of other components in the system. It is an aspect significant for Apollo because it iterates several generations and has improved itself so many times.

However, the disadvantage of Pub-sub Architecture is when a component announces an event, it has no idea what other components will respond to it, and it cannot rely on the order in which the responses are invoked, and it cannot know when responses are finished.

For the Apollo alternative architecture, we believe it can be layered architecture. We have talked about layered architecture in conceptual architecture, which is simple and easy to learn and implement in comparison. It also reduces dependencies because the functionality of each layer is separated from the other layers. For autonomous driving technologies requiring a lot of testing and debugging, layered architecture testing is easier because the components are separated, and each component can be tested individually. What's more, the cost overhead is relatively low.

9. Naming Conventions

ADC Trajectory: Continental Division for Driver Assistance

GPS :Global Positioning System

CANBus : Controller Area Network

HD map: High definition map

HMI: Human–machine interface

Lidar: stands for Light Detection and Ranging, is a remote sensing method that uses light in the form of a pulsed laser to measure ranges (variable distances) to the Earth.

10. Conclusion

To come up a better understanding of the Apollo architecture, we dive deeper into the main system by using Understand to analyze the top level architecture of monitor subsystem, and second level subsystem On-Plane Planning in the Planning module, we figured out some excited findings of discrepancies between the conceptual architecture and concrete architecture. Three unexpected dependencies are discussed in the Monitor subsystem: data subsystem depends on the common module inside monitor subsystem; common subsystem depends on four specific subsystems in common module that are monitor log, util, configs, and adapters; the monitor manager among the common module depends of the 3D rendering of monitored messages in DreamView module. The second level subsystem architecture of the On-Plane Planning is analyzed, the result came out as its RunOnce component depends on the InitFrame component, and the plan component depends on the RunOnce component.

Overall, these findings seem to fit into our devised conceptual architecture style of Publish-Subscribe, since almost every component subscribes to the topic on which they depend. Two sequenced diagrams are proposed for the presented concrete architecture, the first is the Apollo smart transportation, the second is the Apollo’s traffic light perception, where the general use case of Apollo smart transportation is appended by Routing, Task manager, Storytelling, and DreamView modules. Another use case of the traffic light is very similar as we proposed before, the driving decisions are only sent to the control module after the collaborative processing from the planning, prediction, and perception modules. By the end of this report, we have learned how to navigate the code analyzing tool of Understand and utilize it to figure out some cool dependencies between the Monitor module and other modules. However, further detailed analysis may still need to be carried on, since numerous minor dependencies need to be investigated.

11. Lessons learned and Team Issue

According to the investigation on the concrete architecture of Apollo platform, some limitations cannot be controlled by our group. When some of us open the Understand, the tool crashes frequently and responds extremely slowly, moreover, lots of error messages are shown. We emailed Professor Bram and the Head TA, but the issue is still persistent in some of our group members. Meanwhile, detailed dependencies are not investigated since we do not have enough time to look at all dependencies that may have more than 10000 relationships.

Also, from the investigation process, we learn how to use the Understand tool to analyze the code and create the dependency graph that can get a better understanding of the interaction of each module of code. Also, although there are many different parts of Assignment 2, each part has connections with others. In this case, communication is vital since we must change our ideas through communication. Finally, we find that there is still some deficiency since we ignore numerous small dependencies. Although it may not affect significantly, the small dependencies still have their own effect on the whole platform. Therefore, further detailed analysis needs to be conducted.

12. Reference

1. *Apollo Auto*. (2021, December 28). Apollo V7.0.0. https://github.com/ApolloAuto/apollo/releases
2. *Apollo Planning Module Source Code Analysis*. (2018, January 26). CSDN. https://blog.csdn.net/davidhopper/article/details/79176505?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164771953616780261981270%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164771953616780261981270&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-1-79176505.142%5Ev2%5Epc_search_result_control_group%2C143%5Ev4%5Econtrol&utm_term=Apollo%2BPlanning&spm=1018.2226.3001.4187
3. *Apollo Planning Section*. (2021, November 15). CSDN. https://blog.csdn.net/sinat_39307513/article/details/121284148
4. *Apollo Introduction of Planning Module (IV)*. (2021, June 5). Zhihu. https://zhuanlan.zhihu.com/p/61982682
5. US Department of Commerce, N. O. and A. A. (2012, October 1). *What is Lidar*. NOAA's National Ocean Service. Retrieved March 21, 2022, from https://oceanservice.noaa.gov/facts/lidar.html#:~:text=Lidar%2C%20which%20stands%20for%20Light,variable%20distances)%20to%20the%20Earth.