GROUP 24 Diana Candy: Concrete Architecture Report of Baidu Apollo

Linus Zuo (zuo.xinghao@queensu.ca)

Peiyang Huang (18ph18@queensu.ca)

Shengqi Yuan (18sy52@queensu.ca)

Alex Wang (18tw7@queensu.ca)

Yifan Zhu (18yz153@queensu.ca)

Shen Ye (18ys121@queensu.ca)

CISC 322

Software Architecture

March 20, 2022

Contents

1 Introduction

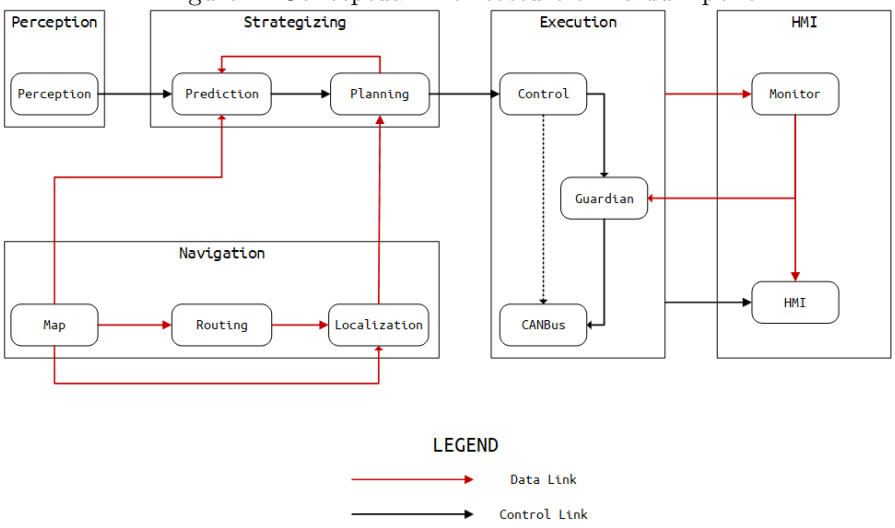
Apollo is an open, complete and secure software platform provided by Baidu to partners in the automotive industry and in the field of autonomous driving, helping them to quickly build their own complete autonomous driving system by combining vehicles and hardware systems. In 2022, it has become the the highest market share in China among all open source autonomous driving systems.

The goal of this report is to determine the concrete architecture of Apollo and perform a reflexion analysis between it and the conceptual architecture we previously derived in Assignment 1. After discussion and research deeply into the code and the dependencies among each modules using Sci-tools Understand, we change the conceptual architecture from Pipe-and-Filter (as we worte in Report 1) to Pub-Sub Architecture. The sub-systems and module inside will still remain the same as report 1.

Along with sharing our derivation process and analyzing the concrete architecture and the discrepancies between it and the conceptual architecture, we will provide further detail into the concrete architecture within the Navigation sub-system. Following that, we will describe two Use Cases and how they interact with the various modules. We will discuss the lessons we have learned from this project.

2 Conceptual Architecture

Figure 1: Conceptual Architecture of Baidu Apollo



2.1 Review of Subsystems

1. ***Perception***: Perception subsystem contains 1 module:

   (a) Perception: The perception module identifies the world surrounding the autonomous vehicle. There are two important function inside

perception: obstacle detection and traffic light detection. The perception module work with 5 cameras (2 front, 2 on either side and 1 rear) and 2 radars (front and rear) along with 3 16-line LiDARs (2 rear and 1 front) and 1 128-line LiDAR to recognize obstacles and fuse their individual tracks to obtain a final tracklist. The obstacle detection submodule detects, classifies and tracks obstacles. It also predicts obstacle motion and position information (e.g., heading and velocity). The traffic light submodule detects traffic lights and recognizes traffic lights.

2. ***Strategizing***: Strategizing contains 2 main modules: Prediction and Planning

   (a) Prediction: The prediction module predicts the future motion trajectories of the perceived obstacles. It works with the obstacles data from the perception module, the localization data from the localization module and the planning trajectory of the previous computing cycle from the planning module

   (b) Planning: The planning module will provide a collision-free and comfortable trajectory. It works with the data from the localization module, Perception module, Prediction module, HD Map from the map engine module, routing module and task_manager. The planning module takes the prediction module result. The planning module plans the route based on that information. Also, the planning module takes the output from routing.

3. ***Excecution***: Excecution Sub-system contains 3 main modules: control, guardian, CanBus.
   The control module takes the route from the planning module and generates the control command (steering, throttle, brake) to pass to CanBus. Also, the Guardian module will prevent the control from sending the command to CanBus when there is some failure. The Control module can work both in normal and navigation modes.

4. ***Human-Computer interaction***: Human-Computer interaction Subsystem contains 1 module: HMI
   Human Machine Interface (H.M.I.) is a web application. It visualizes all the information from each module and allows drivers to check the status of all the modules. It takes the messages from Localization, Chassis, Planning, Monitor, Perception, Prediction. It helps developers visualize the output of relevant driving modules. It is a dynamic 3D rendering of the monitored messages in a simulated world. Developers can send some commands to Guardian to stop the command from Control to CanBus.

5. ***Navigation***: Navigation Sub-system contains 3 main modules: Map, Localization, and Routing

   (a) The map engine queries the map data from the cloud service and provides those data to Localization and other modules. The map engine is also responsible for collecting data from the environment.

(b) The localization module provides localization services. There are two ways to provide localization services. First, the Global Positioning System and Inertial Measurement Unit work together to provide localization services. Second, The Global Positioning System, Inertial Measurement Unit and Light Detection And Ranging Sensor work together to provide localization services.

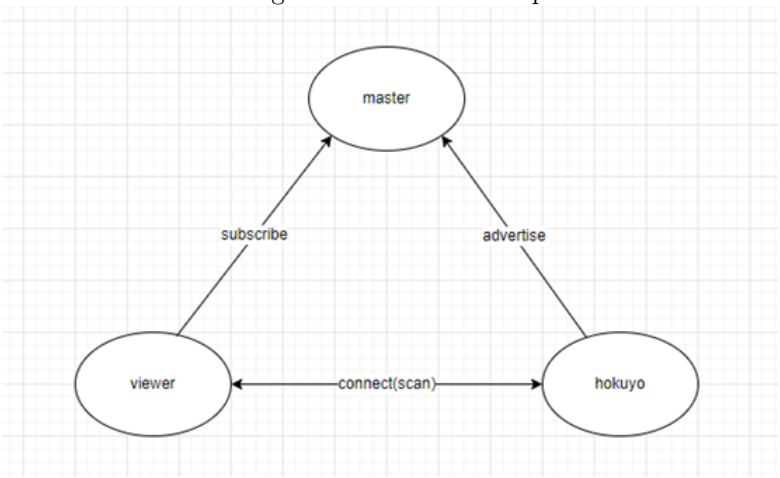
(c) The routing module provide the function as a navigation router.

3 Derivation Process

During our system analysis, our team thought in the last report that Apollo's main system architecture is pipe and filter, but this is not correct; According to Professor's instruction we know that the Publisher-Subscriber architecture is the correct one. So we started from the basics and re-derived and deduced the analysis process. During the analysis of the top-level concrete subsystem, we found that the Apollo project uses the ROS operating system, which is a set of open-source software libraries and tools, but modified on the basis of ROS (in a ROS system, including A series of independent nodes (nodes). These nodes communicate through the publish/subscribe message model, here we confirm that Apollo uses the Publisher-Subscriber architecture).

In the subsequent process of analyzing the Node structure of the Apollo system, we found that one of the Apollo project's transformations of ROS is to remove this centralized network structure. In an interview with Apollo's developers, we learned that Apollo uses RTPS (Real-Time Publish-Subscribe) service as a wire protocol for Data Distribution System.

Figure 2: Pub-Sub transport in ROS



(One big point is that Apollo used CyberRT system to replace part of ROS or cooperate with ROS system) Through further analysis of the node part, we found the upper topic of the node structure. The topic is named buses, which will guide the nodes to exchange messages through the publish-subscribe communication channel. The message should have a point be routed by the ROS network. The node is publishing a topic. Nodes can receive topics from

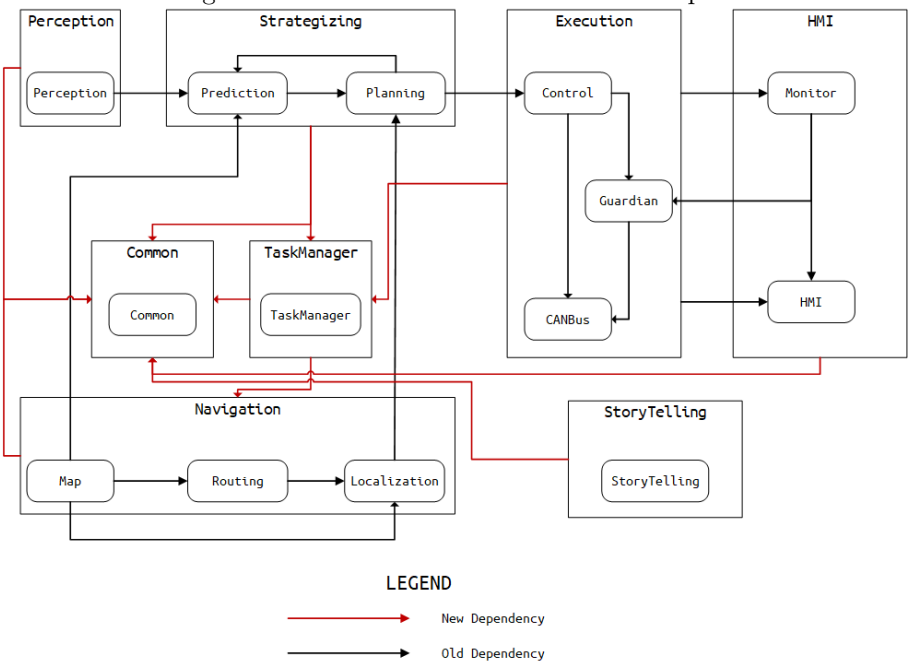
other nodes by subscribing to topics. This further proves that the structure of Apollo is a pub substructure.

4 Concrete Architecture: whole system

Running all our derivation processes, now, we are adding 3 more sub-systems into the concrete architecture of : Task Manager, StoryTelling, and Common. And there are more dependencies detected.

After these new dependencies and subsystems detected, we concluded that the Baidu Apollo System is a Sub-Pub Style system, not a Pipe'n'Filter Style system we reviously believed. This is because, newly discovered subsystems and dependencies suggests that, the whole system works in a Pub-Sub Systle through the new found StoryTelling sub-system which support Pub-Sub features and play a important roles for all sub-systems.

Figure 3: Concrete Architecture of Baidu Apollo



4.1 New Sub-systems

1. ***Common***

   The common module contains code that does not belong to any module. Those codes are very useful in the Apollo system. It interacts with multiple modules and includes different parts such as apollo app, log, macro, adapters, configs/data, math, monitor, proto, status, time, util and vehicle state. It provides the basic way for every module to communicate. Sub-modules in Common sub-system:

(a) **adapters**: Topics are used by different modules to communicate with one another. A large number of topic names are defined in 'adapter_gflags'.

(b) **filters**: implements some filter classes including DigitalFilter and MeanFilter.

(c) **kv_db**: a lightweight key-value database to store system-wide parameters.

(d) **latency_recorder**: record the latency between two time points.

(e) **math**: implements a number of useful mathematical libraries.

(f) **monitor_log**: defines a logging system.

(g) **proto**: defines a number of project-wide protocol buffers.

(h) **status**: is used for determining whether certain functions were performed successfully or not. If not, status provides helpful error messages.

(i) **util**: contains an implementation of a factory design pattern with registration, a few string parsing functions, and some utilities for parsing protocol buffers from files.

(j) **vehicle_state**: specifies the current state of the vehicle (e.g. position, velocity, heading, etc.).

2. ***StoryTelling***

   Storytelling is a global and high-level Scenario Manager to help coordinate cross-module actions. In order to safely operate the autonomous vehicle on urban roads, complex planning scenarios are needed to ensure safe driving. This module creates stories which are complex scenarios that would trigger multiple modules' actions. The main function of this module is to tune the driving experience and also isolate complex scenarios, collecting information from different modules, packaging and publishing them into stories that can be subscribed to by other modules.

   This module provide the ability for every module to subscribe the information public by other modules and use it for themselves. Which makes the whole system a Pub-Sub Style architecture.

3. ***TaskManager***

   Task manager is used to managing the task and different routing problems. It interacts with the map module and common module in this subsystem.

4.2 New Dependencies

1. **[All Kown Sub-System] → Common**

   All known sub-systems will communicate with the Common sub-system. The reason is that, the Common module plays a role of tool chain, provide the ability for every module to get the information from the system and communicate with other modules

2. **TaskManager → Navigation**

   The TaskManager will communicate with modules in Navigation sub-system, it provides the ability to dealing with some special navigation scenarios.

3. **Strategizing → TaskManager**

   As mentioned before, TaskManager provides the ability to dealing with some special navigation scenarios. Strategizing need to use its information to create acurate plan.

4. **Excecution → TaskManager**

   As mentioned before, TaskManager provides the ability to dealing with some special navigation scenarios. Excecution need to use its information together with the command from Excecution to excecute more accurately.

5. **StoryTelling → Common**

   The StoryTelling sub-system will need to collect information from multiple modules in the Common sub-system. These are the modules StoryTelling will communicate:

   (a) **StoryTelling → Common/adapters**:
   Use the topic provide by common to communicate with other modules.

   (b) **StoryTelling → Common/kv_db**:
   StoryTelling will use this to gathering and store information with the database.

   (c) **StoryTelling → Common/latency_recorder**:
   StoryTelling will use this module to collect latency information.

   (d) **StoryTelling → Common/onitor_log**:
   StoryTelling will use this module to access system logs.

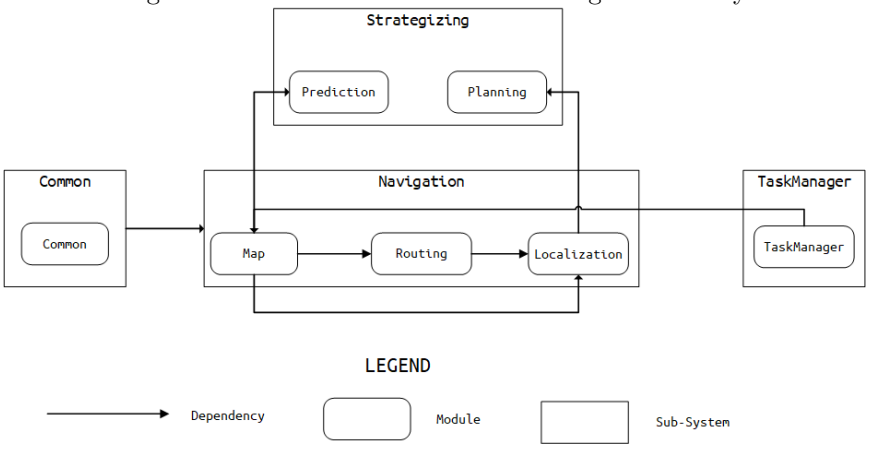
   (e) **StoryTelling → Common/status**:
   StoryTelling will use this module to collect the current status of every functions to see if they are working correctly.

   (f) **StoryTelling → Common/vehicle_state**:
   StoryTelling will use this module to collect the current state of the vehicle (e.g. position, velocity, heading, etc.).

5 Concrete Architecture: sub-system

The Navigation sub-system was investigated to get deeper understand its dependencies. This subsystem was not investigated in the previous report.

Figure 4: Concrete Architecture of Navigation Sub-system



5.1 modules

1. **Map**:
   The map engine queries the map data from the cloud service and provides those data to Localization and other modules. The map engine is also responsible for collecting data from the environment.

2. **Localization**:
   The localization module provides localization services. There are two ways to provide localization services. First, the Global Positioning System and Inertial Measurement Unit work together to provide localization services. Second, The Global Positioning System, Inertial Measurement Unit and Light Detection And Ranging Sensor work together to provide localization services.

3. **Routing**:
   The routing module provide the function as a navigation router.

5.2 Dependency

1. **Map → Routing**:
   Map module will provide information for Routing Information in order to provide correct map for Routing module to design route plan.

2. **Routing → Localization**:
   Routing and Localization will communicate with each other to update current location and make route plan

3. **Map → Localization**:
   Map will communicate with Localization in order to provide correct correct location information.

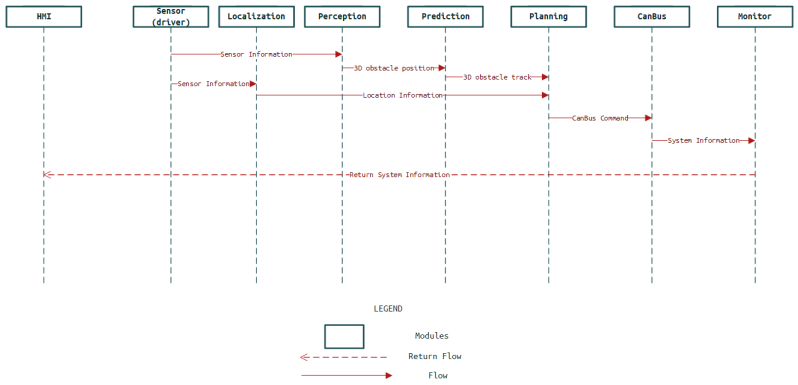
4. **TaskManager → Map**:
   The TaskManager will communicate with map in order to deal with scenario like parking and dead end.

5. **Map → Strategizing/Prediction**:
   Map module will provide information to Prediction in order to predict incoming road event.

6. **Localization → Strategizing/Planning**:
   Localization will provide information to Planning in order to make accurate driving plan.

6 Use Cases

1. **Automatic obstacle avoidance**:



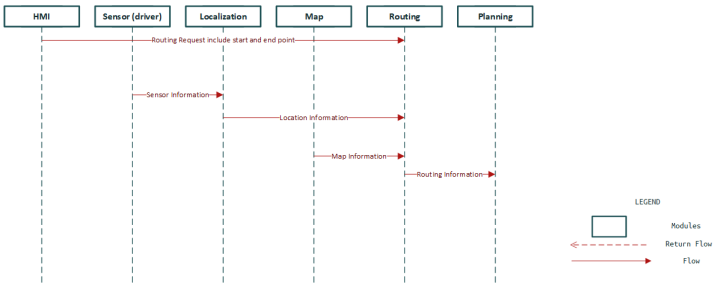
   For the use case of automatic obstacle avoidance use case, it uses the following steps:

   Firstly, the native sensor information will be collected by the driver module, and it will be transmitted to the perception module for processing. The perception module will process the native sensor information and generate the 3D obstacle information (by the sensor) data.

   Then the 3D obstacle information data will be transmitted into the prediction model. In prediction model will analyze the 3D obstacle information (by the sensor), and calculate the prediction. When the prediction module has done its part, it will generate the 3D obstacle track. This information will be sent to the planning module.

   The planning module will collect data from multiple different modules, in the obstacle avoidance use case, it will collect the 3D obstacle track from prediction and car's location information from the localization module. The planning module will collect this information and calculate how the vehicle will travel next as the CanBus command. This command will be transmitted to the CanBus module and be executed by it.

2. **Navigation route planning**:



The use case of navigation route planning use case uses the following steps:

In the beginning, the navigation request will be sent from the H.M.I. module by the user, after that the routing module will begin to wait for the information of location and map.

The location information will be first collected by the sensor (driver module), after that, this information will be sent to the localization module. The localization module will calculate the location data from the sensor data, after it has done the calculation, the location data will be sent to the routing module.

On the other hand, the map will send its information to the routing module together with the location data calculated by the location module. After this data is collected, the routing module will begin to calculate the navigation route plan. After the routing module is done, this information will be sent to the planning module and wait for planning and execution.

7 Current Limitations and Lessons Learned

When we analyzed the code, we visited Apollo's Chinese developer community and interviewed some partners and developers. They said that the development of pub-sub in CyberRT mode, In other words, CyberRT also has the shortcomings of less user experience, and the resources are not as comprehensive as ROS. In some parts, some team-based development models make up for these problems. Communication in groups is sometimes more effective, and it is responsible for the detailed division of labor in different parts of the specific architecture to improve the problem effectively.

Software development is an iterative process, and many Apollo partners said that tracking the latest structural changes is a critical point in their work, so to serve these partners reasonably, services such as pub-sub have scalability, Availability, and latency advantages are very important, which were pointed out at the beginning of the Apollo program. Any asynchronous messaging service has to consider these issues from the start, and we learned to focus more on building and communicating from the beginning of the plan.

During this analysis, our group conducted more frequent group exchanges, held three meetings a week, and frequently updated the reference materials we found in the communication software, which give help to each of us with the task of completing our assignment, everyone can learn about hidden spots that they overlooked. And check the omissions and fill in the vacancies and communicate with each other the content that was not understood or misunderstood during the class

8 Conclusion

After spending a lot of time with Sci-tools Understand and the source code, our team can now take a deep look into the system without the limitation we faced in the first report. Now, we are adding 3 more sub-systems into the concrete architecture of the system: Task Manager, Story telling, and Common and multiple new dependencies. We also analyze the dependecies and modules of one sub-system. Now we have a much deeper understanding of how Apollo is constructed

9 Reference

1. Apollo Development Team. 2021. Baidu Apollo System. [Source Code] https://github.com/ApolloAuto/apollo
2. Apollo Development Team. 2021. Baidu Apollo System. [Changelogs] https://github.com/ApolloAuto/apollo
3. Apollo Development Team. 2021. Baidu Apollo System. [Documentations] https://github.com/ApolloAuto/apollo