CISC 322 A2 Presentation

Adam Cockell, Ashton Thomas, Udbhav Balaji, Dahyun JIN, Udit Kapoor, Kevin Subagaran

Presentors: Adam Cockell and Ashton Thomas

Introduction - 1

- Our previous report explored the Apollo open source project to discuss the various components and interactions within the system
- In this report we aim to determine the concrete architecture of Apollo and perform a reflexion analysis between it and the conceptual architecture discussed in the report from Group 6
- This analysis is supported using SciTools Understand to extract code dependencies within the system and identify an optimized concrete architecture
- We will discuss any discrepancies between the conceptual and concrete architectures discovered in reflexion analysis and make changes to the conceptual architecture in order to minimize unexpected dependencies and better match the concrete architecture.

Introduction - 2

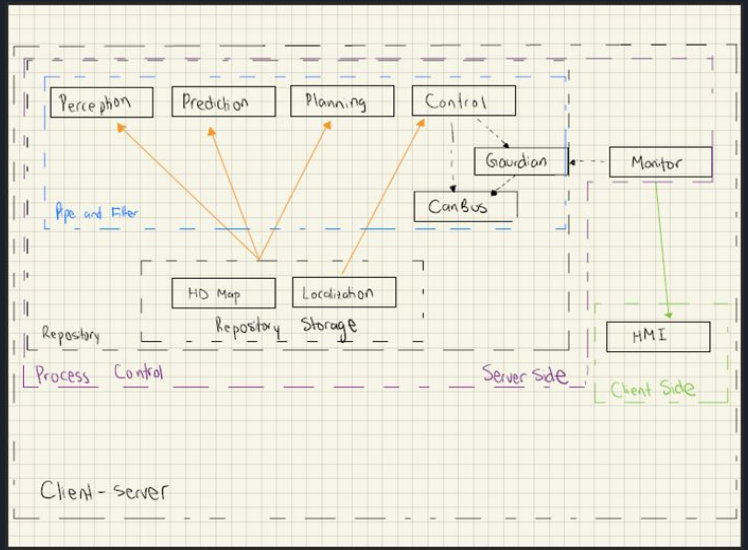
- Aside from reflexion analysis, we will discuss the addition of two new components – Monitor and Guardian – and how they integrate with the existing components in the system, with further analysis provided on one of the new components
- Following that, we will cover two use cases in the context of the extracted concrete architecture, and how the Apollo system works to solve a given problem
- Finally, we will discuss issues our team encountered in researching and writing and conclude with lessons learned while working on this report.
- In this report, we will combine our previous assessment of the Apollo architectural style with that included in the report written by group 6

Review of Subsystems

- Map Engine: This module provides high-fidelity map data - using the HD-map component - to the next 4 modules.
- Localization: This module provides high-accuracy positional data based on GPS, IMU, and other systems to the next 4 modules.
- Perception: This module detects obstacles and the general surroundings of the vehicle using cameras, radar, and LiDAR. This information is sent to the Prediction and Planning modules.
- Prediction: This module uses current velocity data of detected obstacles to calculate where they will be in the future. This information is then sent to the Planning module.
- Planning: This module is responsible for choosing the path of least risk given all the information from the prior modules. Additionally, the chosen path must abide by traffic laws, and be comfortable to navigate for any passengers on board as the chosen path is followed by the Control module.
- Control: This module takes input from the Planning, Localization and Dreamview/HMI modules to output vehicle control commands.
- HMI: This module provides a web-application that helps developers visualize the output of other relevant autonomous driving modules. This can include driving trajectories, car localization, and more.
- Monitor: This module monitors the other modules to verify that all the systems are functioning normally. This includes hardware resource usage, data integrity, and latency metrics.
- Guardian: This module secures the entire system in case of failure in a component. When a failure is detected, control flow is stopped so that potentially dangerous commands are not issued to the vehicle.

Conceptual Architecture - 1

- The pipe-and-filter, repository, process control, and client-server styles are used in combination to construct this model
- The pipe-and-filter style is the primary style that allows Apollo to dynamically produce control commands for the vehicle
- Perception, prediction, planning, and control modules are the filters, and the pipe is the control or data flow between them.

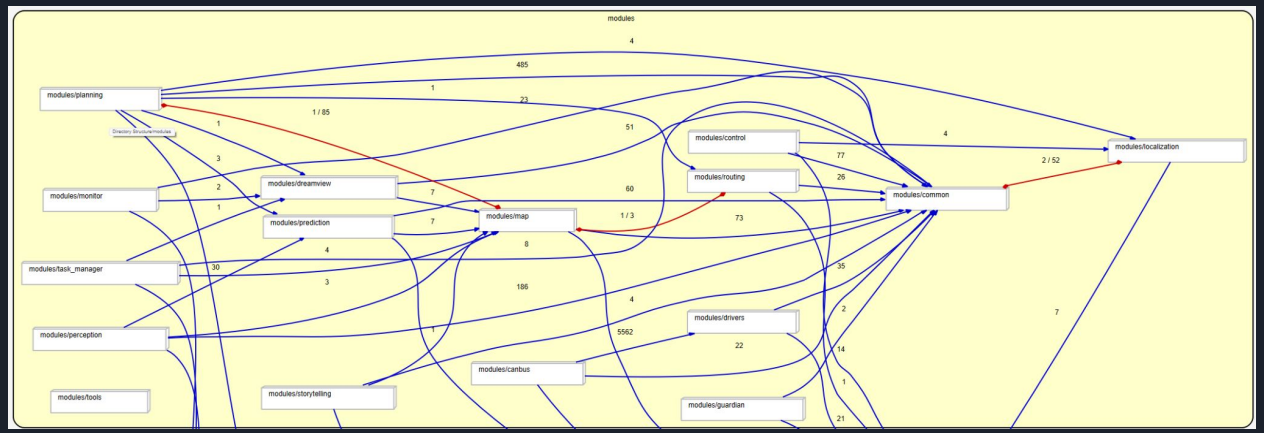


Conceptual Architecture - 2

- The repository style sits on top of the pipe and filter styles. The map engine and localization together form a central data repository that delivers data to the perception, prediction, planning, and control modules
- Meanwhile, a process control style is maintained with the inclusion of two new components: monitor and guardian. These two components are responsible for providing a safe driving experience
- Next, Apollo has a client-server design - HMI is a client that shows the user all relevant data and vehicle status, and is in charge of displaying input data on the stage
- Finally, the PubSub architectural pattern ensures little risk of performance degradation while maintaining low complexity, providing scalability and flexibility

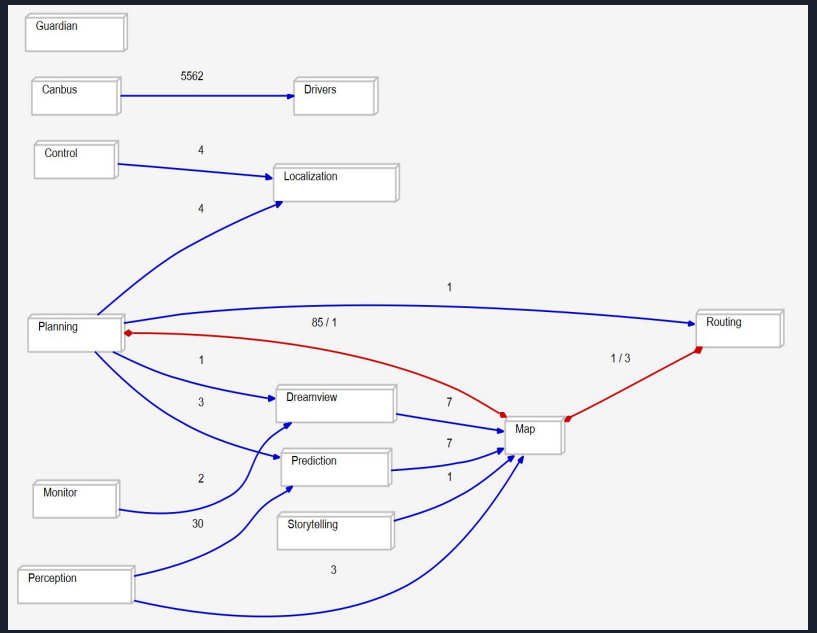
Derivation Process - 1

- To derive the concrete architecture of Apollo, we analyzed the static dependencies in the code base and looked at the pub-sub message traffic between different modules
- We used a tool called SciTools Understand that allows us to visually analyze the dependencies of a code base
- We used an extracted dependency file to visualize the file structure and dependencies of the system



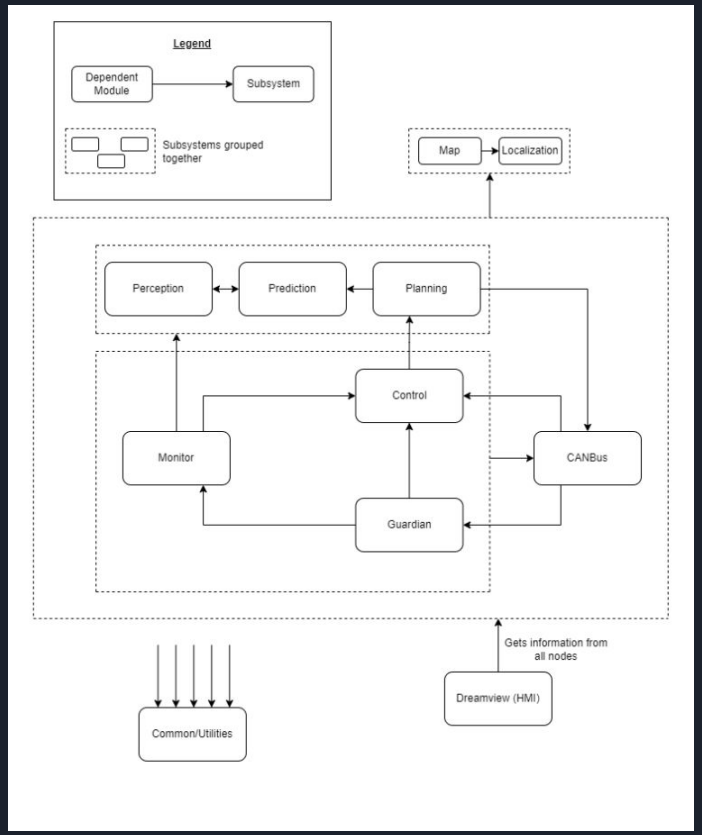
Derivation Process - 2

- We added a common utilities module since it was large and most modules used files from the common module
- We kept simplifying the modules by combining ones that fit together to better match the conceptual architecture
- After reaching this step, we could remove connections with too few dependencies, since almost everything has a dependency in a given module in such a large code base
- Combining this data with the pub-sub message traffic graph, we were able to extract the final concrete architecture of the system



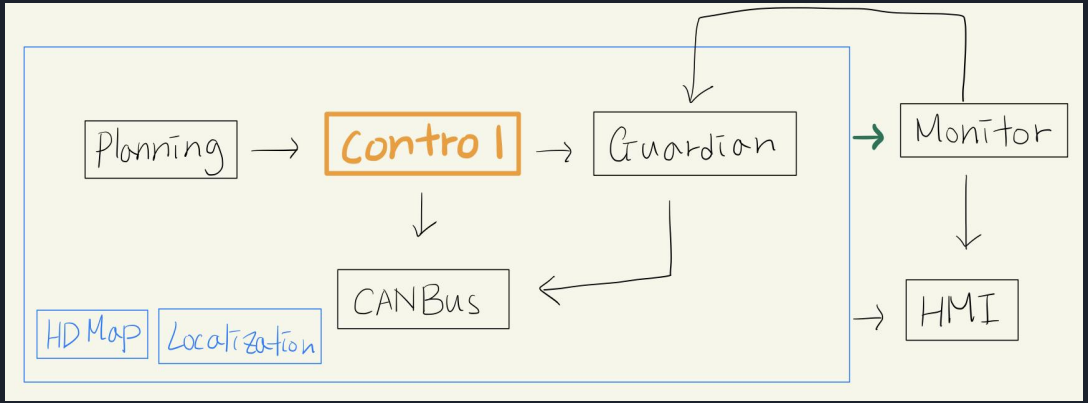
Concrete Architecture - 1

- The software follows a Pub-Sub architecture model, where several modules “publish” their results and data which can be “subscribed” by other modules for their use
- While many modules don’t have static code dependencies on some other modules, they subscribe to their data outputs to function correctly
- We kept most of the subsystems the same as the conceptual architecture, but added a common/utilities module since it was too large to ignore and almost all other subsystems depended on it



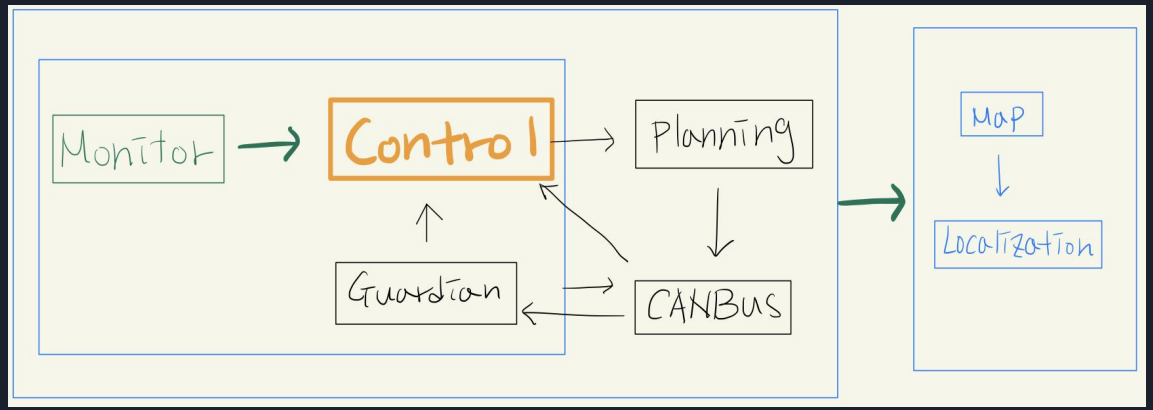
Concrete Architecture - 2

- Control is a subsystem that uses different algorithms for a comfortable driving experience based on the planning trajectory and the car's current status
- Based on Apollo 5.5, Control got new features such as Model Reference Adaptive Control(MRAC) and Control Profiling Service
- The purpose of MRAC is to effectively offset the by-wire steering dynamic delay and time latency
- In addition, it also ensures faster and more accurate steering control actions for tracking the planning trajectory.



Concrete Architecture - 3

- Control Profiling Service provides a systematic quantitative analysis of the vehicle's control performance via road-tests or simulation data
- Control gets inputs such as planning trajectory, car status, localization, and dream view AUTO mode change request while output control commands like steering, throttle and brake to the chassis
- Unlike the conceptual view, in the concrete view, Control has direct connection to monitor, and also Map and Localization are grouped separately from Control



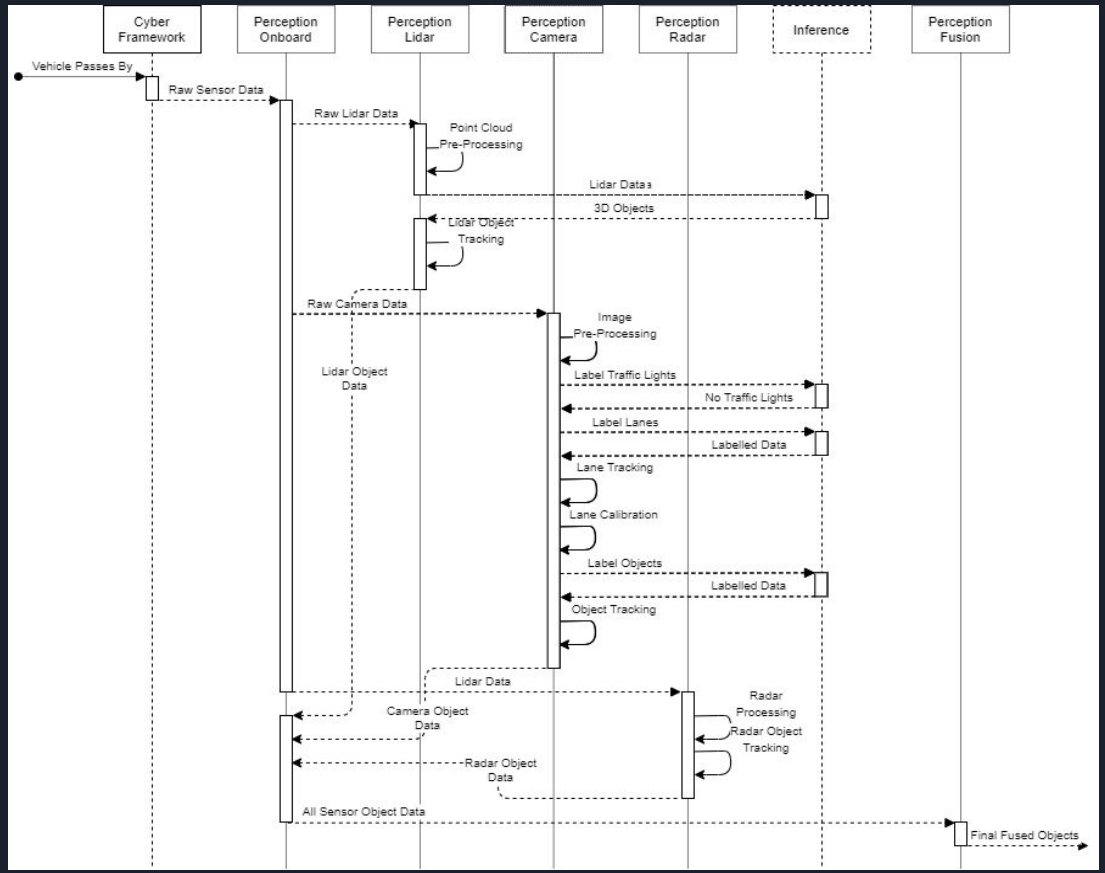
Discrepancies between Conceptual and Concrete Architecture - 1

- A major change that we made to the high-level architecture of Apollo is the addition of the Utilities module, which effectively links all the other modules together
- Along with this, another modification is that modules with similar tasks are grouped together in order to enhance user readability and understanding.
- The Control module also has great features that will be incorporated into the architecture such as MRAC and CPS, which help the system react better to unexpected stimuli in the environment and help make Apollo a safe, stable option on the road
- This includes getting more detailed data regarding car trajectories, as well as improved localization data, helping the car plan a better trajectory in times of need

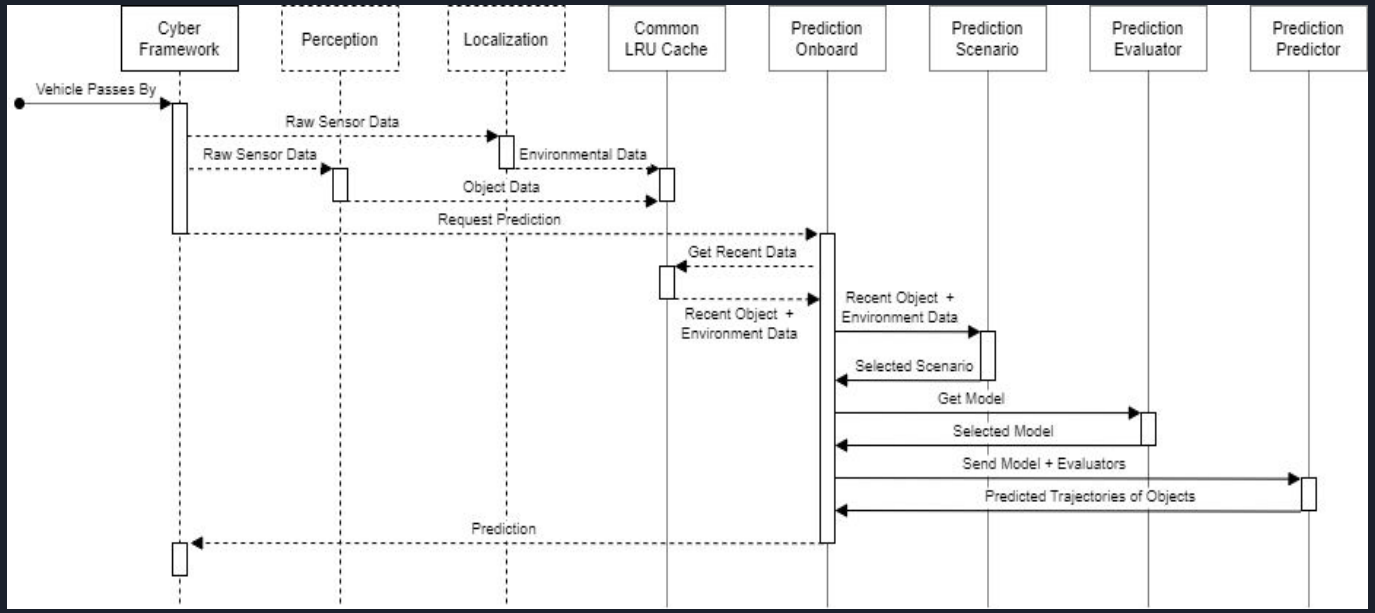
Discrepancies between Conceptual and Concrete Architecture - 2

- In our concrete architecture, a gap that might have to be filled would be the efficiency of data and flow control throughout the system
- Another major change would be regarding the HMI module - In the conceptual architecture, it only has the Monitor module as a dependency, while in the concrete architecture, it has all the modules as dependencies
- Since it takes input from all the other modules, a direct link between the other modules and HMI might improve efficiency but increase the complexity of the architecture itself and how this module works.

Use Case #1 - Perception Module



Use Case #2 - Prediction Module



Lessons Learned

- We learned how to use SciTools Understand to analyze the code dependencies of a codebase and learn more about its architecture
- Apollo is an incredibly complex software that has been in development for almost a decade and it took a lot of reading code, documentation and PubSub relations to understand the concrete architecture

Conclusion

- From the mappings deduced for assignment 1 and analyzing Apollo's codebase using SciTools Understand, we were able to find the Concrete Architecture of the software
- There were significantly more dependencies between modules, likely due to the Concrete Architecture evolving over time from its initial start as the Conceptual Architecture we deduced
- By contrasting the concrete and conceptual sequence diagrams it is apparent that the software uses a pub-sub message system
- This system was chosen as it is the best fit for a real-time operating system, and allows for easy real-time communication between modules and their submodules
- It also suits the system as it streams sensor data directly to the perception and prediction modules
- The concrete sequence diagram also showcases the cyber framework and its communication between modules

THANK YOU!