Openpilot System Architecture Discrepancy Analysis

by: Team Horizon

Authors

Ali Sheikhi	alish@my.yorku.ca
Erika Grandy	egrandy@my.yorku.ca
Jaskirat Pabla	pabla1@my.yorku.ca
Mate Korognai	korognaimat@gmail.com
Nabi Khalid	khalid18@my.yorku.ca
Natalie Dean	natrdean@my.yorku.ca
Shaharyar Choudhry	sharry09@my.yorku.ca
Suha Siddiqui	suhasid@my.yorku.ca

Abstract

This report highlights the conceptual and concrete software architectural composition of the openpilot system that were gathered from previous documentation. In specific, openpilot's overall system architecture and the subsystem known as panda will be discussed in great detail. The conceptual model and the concrete model will be highlighted individually through the use of extracted diagrams that were gathered from documentation, source code, directory structure and the Understand software. Then, comparisons between the conceptual and concrete architectures will be made from the extracted diagrams. Following the comparisons, suggested changes to the conceptual model will be provided with updated diagrams. The insights gained from our comparisons will be used to provide these updated diagrams with the team's rationale for the choices that were made.

There were many discrepancies present between the conceptual and concrete models that were found. As a team, we found that the conceptual model differed from the concrete due to the complexity present in the system. These differences also stem from hidden dependencies that were not discovered due to only looking at online documentation rather than source code for the conceptual model.

The concurrency of the system will also be discussed. It was found that the team partly predicted the concurrency present in the openpilot system. It was claimed that due to the presence of an event bus, concurrency will occur in the communication subsystem of panda. We found that the entire system seems to be using implicit invocation and thus the concurrency is present all across the system.

The lessons learned through both Assignment 1 and Assignment 2 will be discussed. Furthermore, the possibilities for why the discrepancies between the conceptual and concrete model will be looked at. Following this, a conclusion will be given that encompasses everything that the team discovered for Assignment 3.

Introduction and Overview

This report dives into the architecture of openpilot, an open-source project that pushes the boundaries of what's possible in autonomous driving technologies. Previous assignments investigated the architecture of openpilot. Assignment 1 focused on the conceptual composition of the system that was derived from online documentation and the directory

structure of the source code (without actually looking at the source code). Assignment 2 focused on the concrete architecture that was gathered using directory structure, source code and the Understand software. This assignment, Assignment 3, focuses on comparing the extracted information from both Assignment 1 and Assignment 2. The analysis process of how to compare these findings will be in the section *Analysis Process*.

First, the findings from both assignments will be highlighted in the *Top Level of Subsystems and Interactions* section. In this section diagrams of the conceptual and concrete architecture will be provided with explanation of what they highlighted. In the following section, *Comparison of System Architecture*, highlights the differences between each section and proposes some changes that could be made to the conceptual architecture to more closely match the concrete model.

Likewise, the findings from both assignments will be highlighted in the *Top Level of Panda Subsystem* section. This section will focus on providing our understanding of the conceptual and concrete architecture of the panda subsystem using diagrams and explanations. Following this, the section *Comparison of Panda Subsystem*, the difference between our understanding of the conceptual and concrete model for panda will be highlighted and proposed changes will be applied so that the conceptual model matches the concrete model more closely.

The concurrency will then be discussed, and our understanding of the concurrency from both Assignment 1 and Assignment 2 will be compared. Then the lessons learned will be discussed that will highlight the things we have learned and what we would do differently if given the chance to redo the project or refine our conceptual model. Finally, a conclusion of the overall system will be given.

Analysis Process

The analysis of the conceptual and concrete model for both the overall openpilot system and the panda subsystem will be very similar. First, an understanding of the conceptual and concrete architecture must be established using diagrams and explanations so they will be discussed before moving onto the comparison phase. In the comparison phase, the provided diagrams will be discussed and the discrepancies will be highlighted. This is the actual analysis phase and it focuses on approaches to rework the conceptual model to closer match the concrete model through the use of rationale that will be provided.

Top Level of Subsystems and Interactions

Report 1 discussed a high-level architecture of openpilot listing Layered, Implicit Invocation (Event-Based and Publish & Subscribe), Process Control (MAPE-K and Closed Loop Feedback), Client-Server, Pipe and Filter, and Repository as architectural styles used. This version also included information about the subsystem breakdown involved in the functionality of this system such as sensors, actuators, neural network routes, localization, calibration, controls, logging, miscellaneous services and hardware.

A wide variety of components were commented on in terms of their involvement in the architectural styles such as panda, opendbc, boardd, and cereal (figure 6). Furthermore, the report detailed how the subsystems and architectural components are involved in the data

and communication flow in figure 3 (concurrency and simultaneous task execution). Much of our analysis was from various sources available online and previous studies done on the system. Also, we touched on the fact that the system will be bound to evolve due to the open-source nature of the system. Moreover, we claimed that openpilot has many developers involved with various roles to provide effective functionality.

Report 2 concretely approached openpilot by focussing on one subsystem called panda. The Understand software was used to visualize the dependencies and analyze the software in greater detail (figure 1). Also, the components which were discussed in the previous report (i.e. panda, opendbc, board, and cereal) were visualized in Understand to show their interconnectedness (figure 2). After using the tool we observed some architectural styles such as Implicit Invocation which we could confirm from our previous report. Furthermore, the role of the panda subsystem was analyzed in terms of its role in the data and communication flow between the hardware and software components in openpilot. Additionally, external interfaces that were involved in the data and communication flow were discussed such as CAN. Also, we concluded that there was a lack of design patterns used as it is primarily coded in C which is not object oriented. Finally, we concluded the role of panda in the greater openpilot system, how pandas depend on other components, what components depend on panda's, and the importance of security in communication and dataflow.

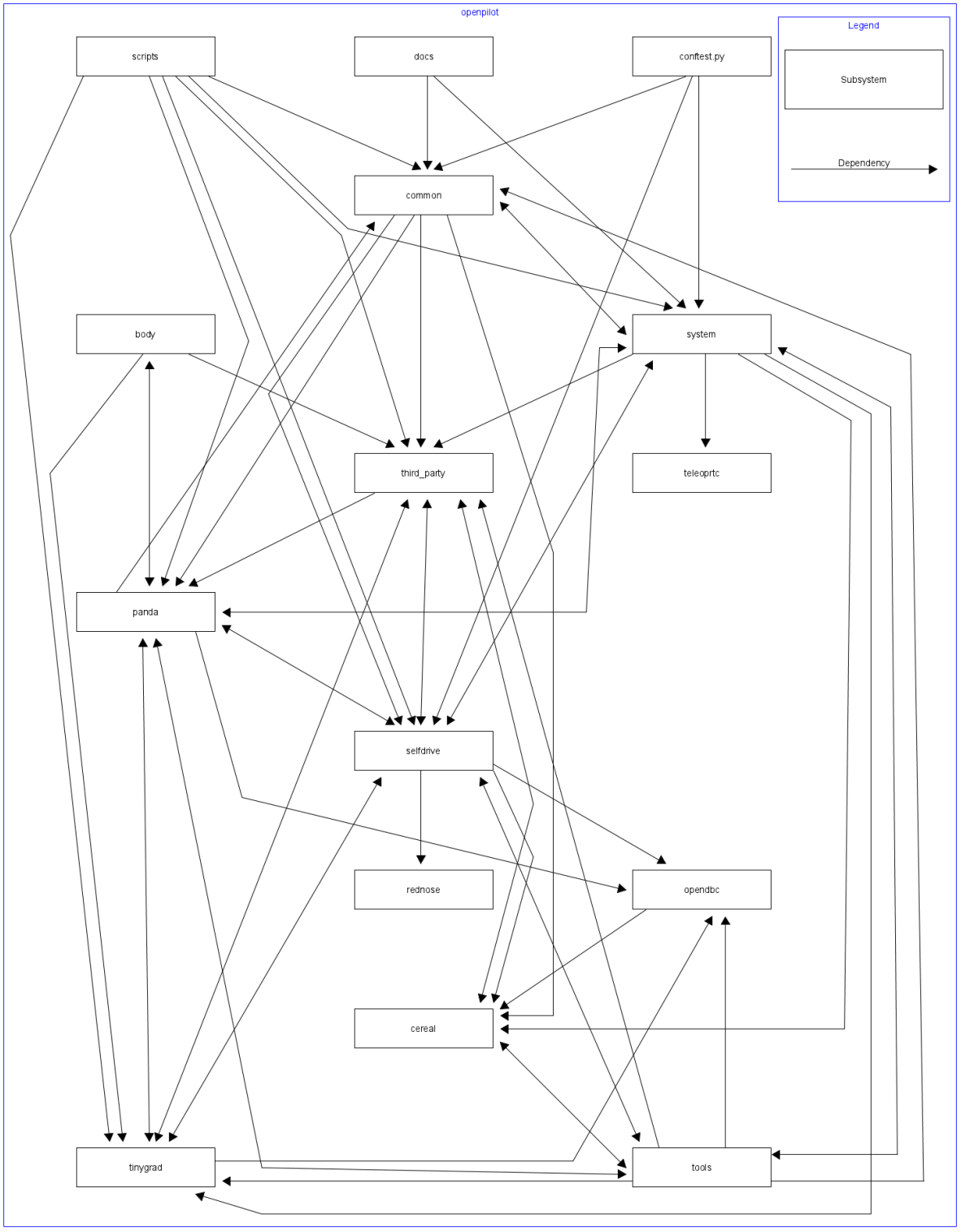

Figure 1: The highest level of subsystem dependency relation of the openpilot system

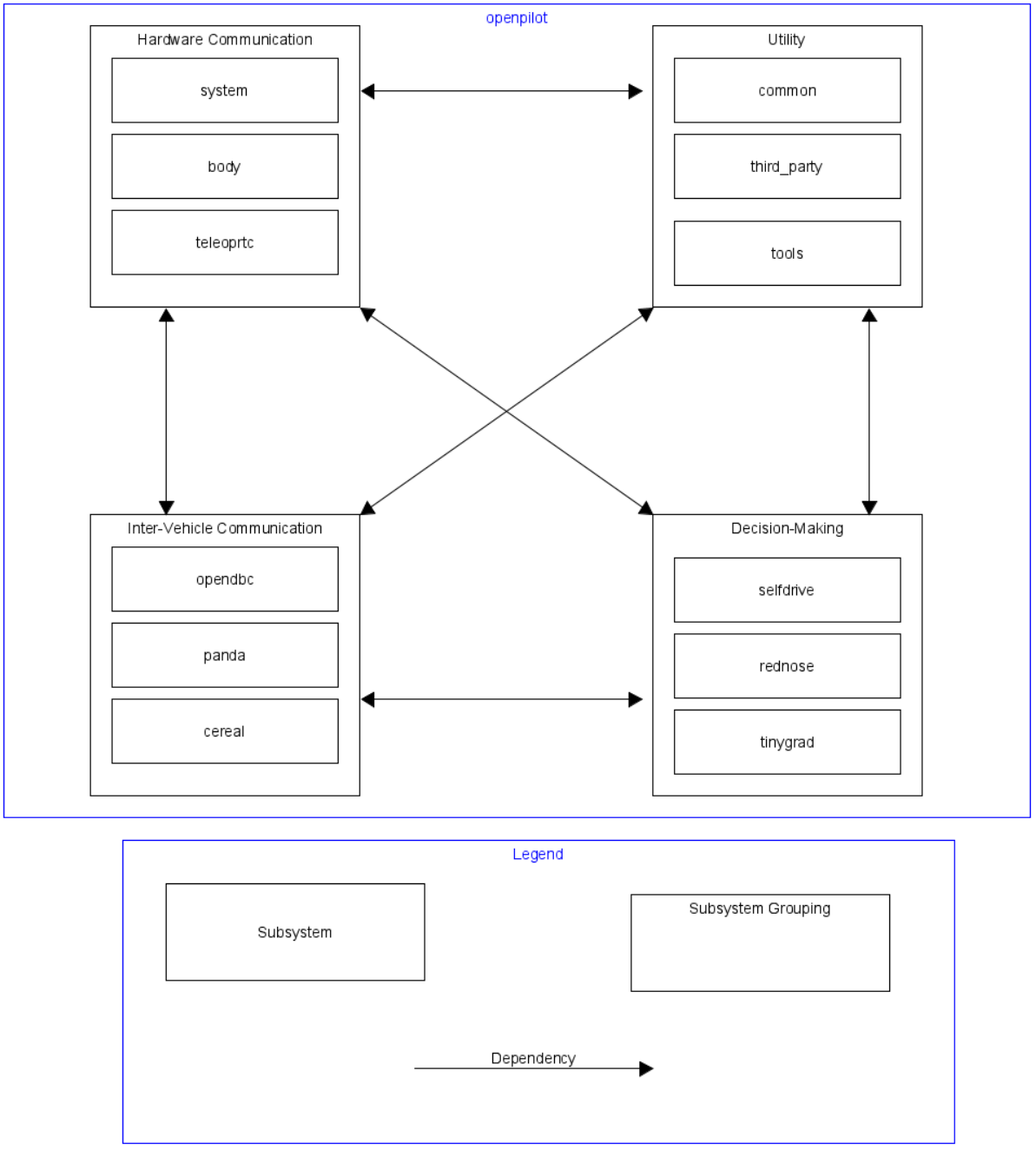

Figure 2: The highest level of subsystem categorization dependency relation of the openpilot system.

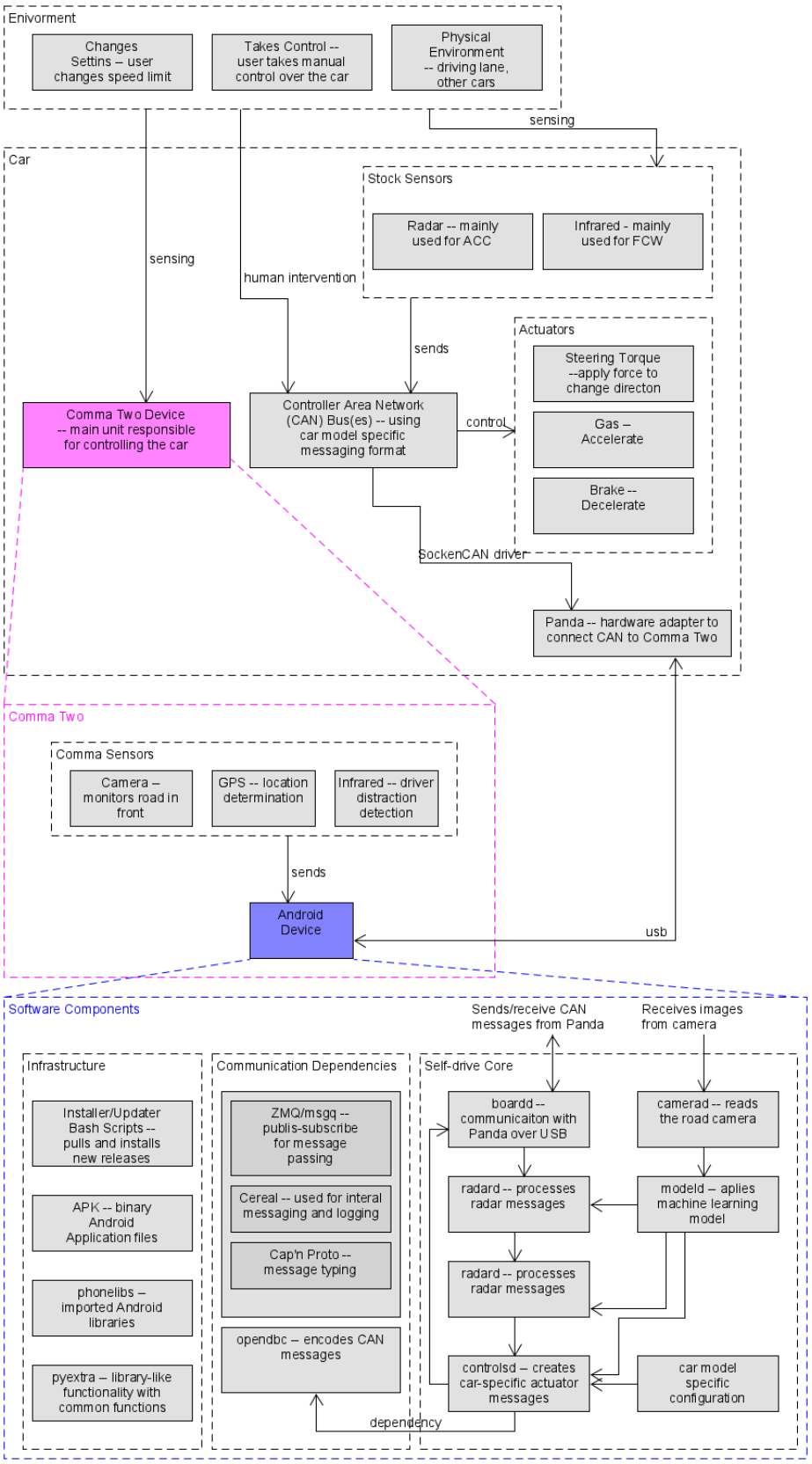

Figure 3: Delft University architectural diagram demonstrating areas of concurrency [1]

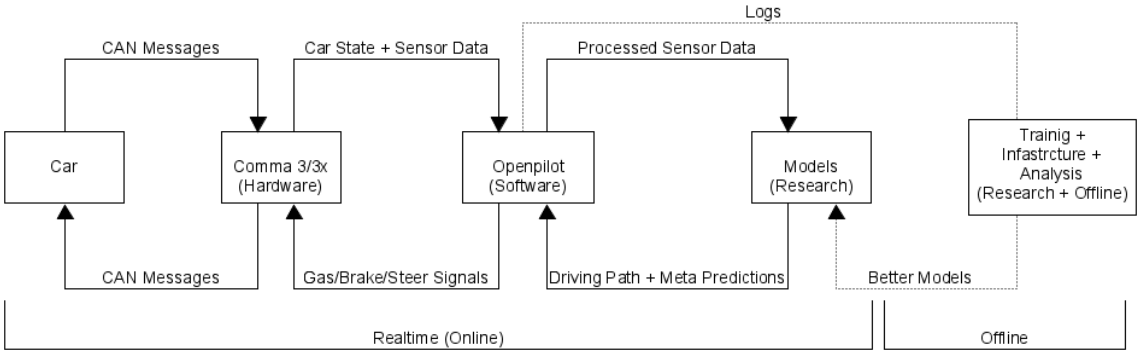

Figure 4: Interaction of technical components and breakdown of parts comprising openpilot 0.9.5 [2]

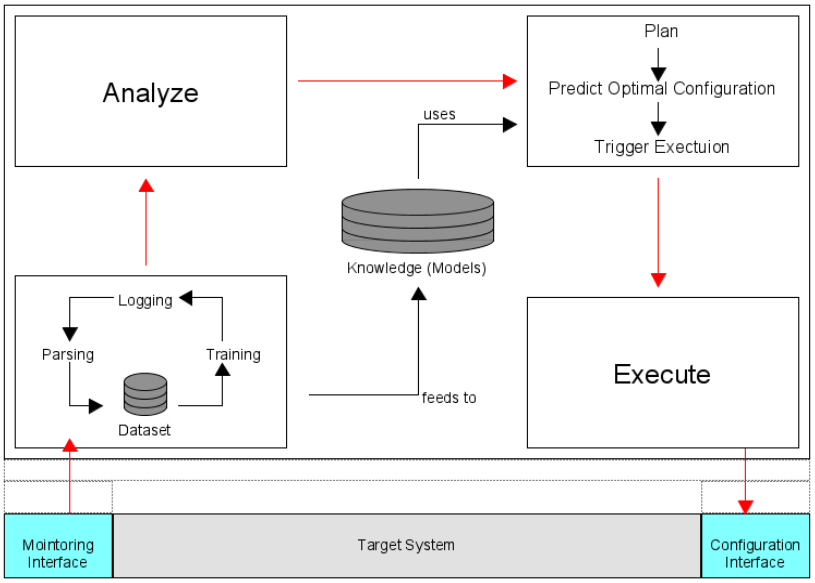

Figure 5: MAPE-K with Machine Learning [3]

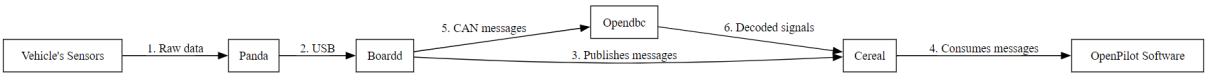

Figure 6: Communication Flow among openpilot components

Comparison of System Architecture

Reflexion Model Analysis:

When comparing the conceptual architecture of openpilot with the actual software architecture reflected in the system's code, several differences stand out. The high-level conceptual model, which was described using terms like MAPE-K and Client-Server, does not fully align with the details found in the concrete implementation. For instance, the expectation that openpilot operates with a MAPE-K loop, which suggests a self-regulating system, is not supported by the findings in the concrete analysis. The concrete details, specifically within the 'panda' subsystem, don't show this loop structure.

Similarly, while the conceptual model initially included a Client-Server architecture, implying a network of independent systems that communicate over a network, the actual dependencies and interactions mapped out in the concrete model (shown in figure 2) do not directly substantiate this architecture. We acknowledge that a Client-Server could be used in the system as a whole, however, in the overall system architecture there seems to be no Client-Server architecture present.

The layered architecture, which conceptually organizes system operations into hierarchical levels (figure 3), seems to blur in the concrete model due to the numerous dependencies and interactions that cross these boundaries. When we count the actual connections in the concrete architecture (observable in figure 1 and figure 2), we find many more than the conceptual model implies, indicating a far more complex and interlinked system than the simplified conceptual layers would suggest.

The Closed Loop architecture is typically used for systems that require continuous control and feedback. It was likely envisioned in the conceptual model of openpilot to maintain system stability and adaptability. However, its absence in the concrete model could be attributed to the shift towards more open, flexible, and possibly decentralized control mechanisms that do not fit the traditional Closed Loop paradigm. This might be due to the complexity of real-world driving scenarios where a rigid Closed Loop system may not be able to accommodate unpredictable variables.

The Pipe & Filter architecture is designed for systems that process a stream of data through a sequence of transformations. This architecture was probably omitted in the concrete model because the data processing needs of openpilot might have evolved to require more dynamic and interactive processing strategies rather than the linear, stateless processing implied by Pipe & Filter. The need for real-time decision-making and complex data handling in autonomous driving systems could have led to the adoption of more sophisticated architectures that provide greater flexibility and integration capabilities.

The Repository architecture is often used to provide a centralized data access layer that abstracts the underlying data storage and retrieval mechanisms. Its absence in the concrete model suggests that openpilot may have moved towards a more distributed data management approach, possibly to enhance scalability, fault tolerance, and performance. The highly distributed nature of sensor data and the need for localized processing at the edge might have rendered a centralized repository impractical for the system’s operational requirements.

These observations indicate that the current conceptual model needs to be updated to reflect the intricacies observed in the concrete architecture. This would involve recognizing the dense network of interdependencies and the real software architecture patterns in use. Adjusting the conceptual framework to mirror these findings will provide a more accurate and functional map of openpilot's system, which is critical for ongoing development and analysis (as the differences noted in figures 1 through 6 show). This revised model should better guide the understanding and enhancement of openpilot's architecture.

In summary, the transition from the conceptual model to the concrete model in openpilot reflects a shift towards architectures that better accommodate the complexities and demands of autonomous vehicle systems. These changes underscore the importance of

flexibility, scalability, and real-time processing in the development of such advanced technologies.

- **Convergences**
  - Implicit Invocation (architecture)
  - panda
  - opendbc
  - cereal
  - boardd
- **Divergences**
  - tinygrad
  - common
  - system
  - teleoprtc
  - third_party
  - body
  - tools
  - rednose
  - selfdrive
- **Absences**
  - Process Control (architecture)
    - Closed Loop
    - MAPE-K
  - Client-Server (architecture)
  - Layered Architecture (architecture)
  - Pipe & Filter (architecture)
  - Repository (architecture)

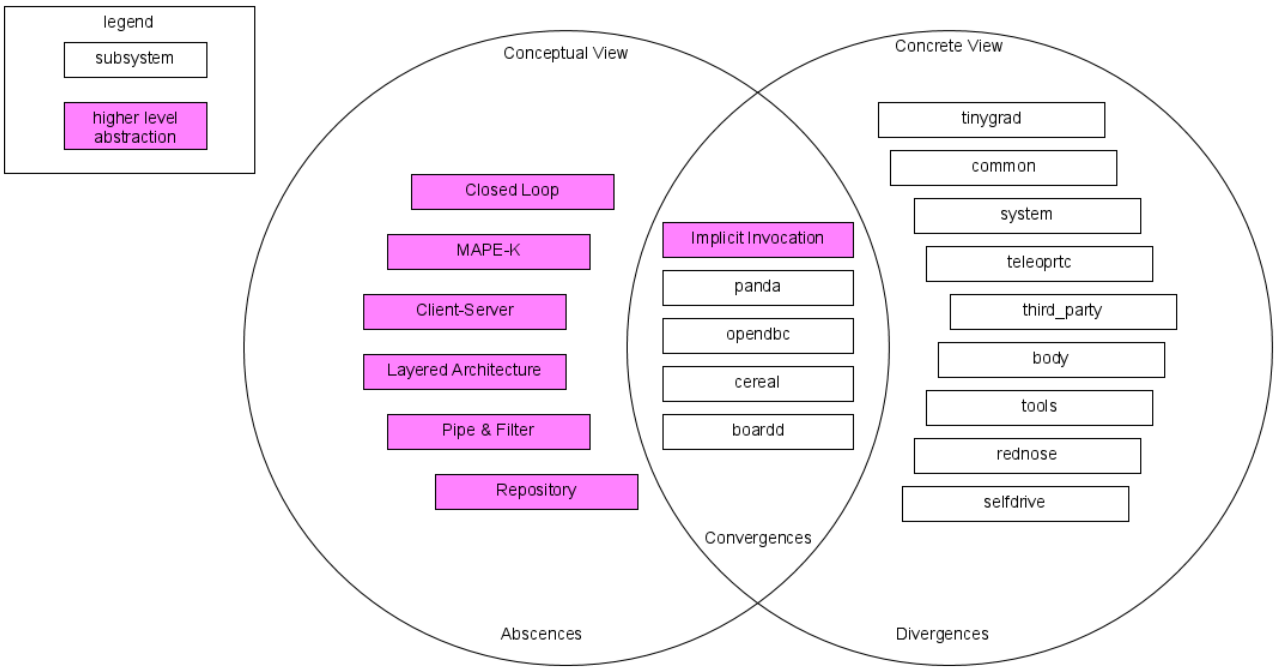

Figure 7: Venn Diagram of Conceptual vs Concrete System Architecture

Top Level of Panda Subsystem

In developing the conceptual architecture surrounding the panda subsystem, we determined that panda had dependencies on the boardd, opendbc, and cereal subsystems and communicated with the vehicle CAN bus, collection of sensors, and collection of actuators. The dependency on boardd was determined from information on boardd facilitating communication to panda from other openpilot subsystems. The dependency on opendbc and communication with the CAN bus was determined from documentation detailing that panda communicates with a vehicle's CAN bus and that this is achieved through dbc files managed by opendbc. The dependency on cereal and communication with the sensors and actuators was determined from information describing how panda communicates with sensors and actuators and documentation detailing that cereal is the facilitator of communication with sensors and actuators.

After extracting the concrete architecture surrounding panda, we found that the subsystem actually had dependencies on the tools, selfdrive, body, tinygrad, system, opendbc, and common subsystems. We observed that boardd was actually a part of panda, not a separate subsystem, and that panda had no direct dependencies on cereal. The majority of the information we found about panda's dependencies when developing the conceptual architecture regarded its role in communication between openpilot and the vehicle. After extracting the conceptual architecture, we were able to see that panda additionally depended on selfdrive and tinygrad for decision-making purposes, common and tools for utilities, and system and body for other hardware communication purposes. We also learned that panda had no direct communication with the CAN bus and collections of sensors and actuators and that this communication was instead primarily managed through opendbc.

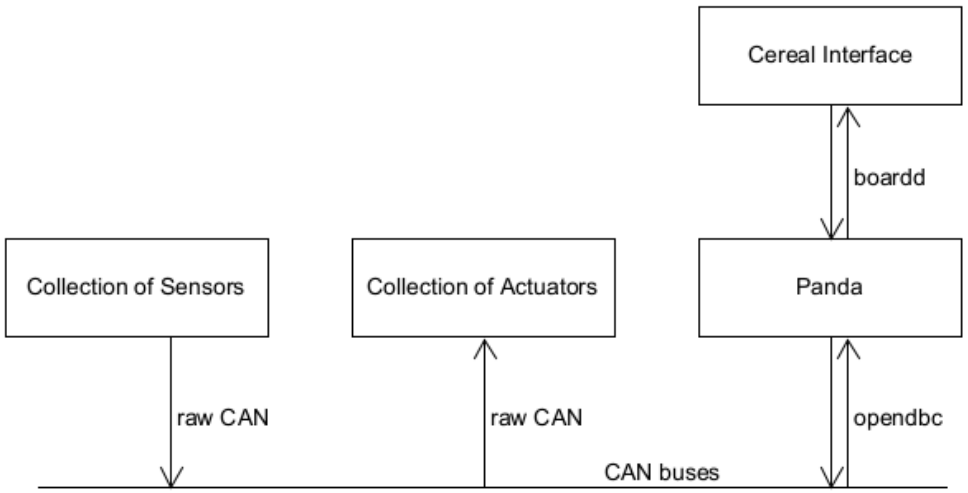

Figure 8: Subsystems and Dependencies Involved in Communication Between openpilot and a Car

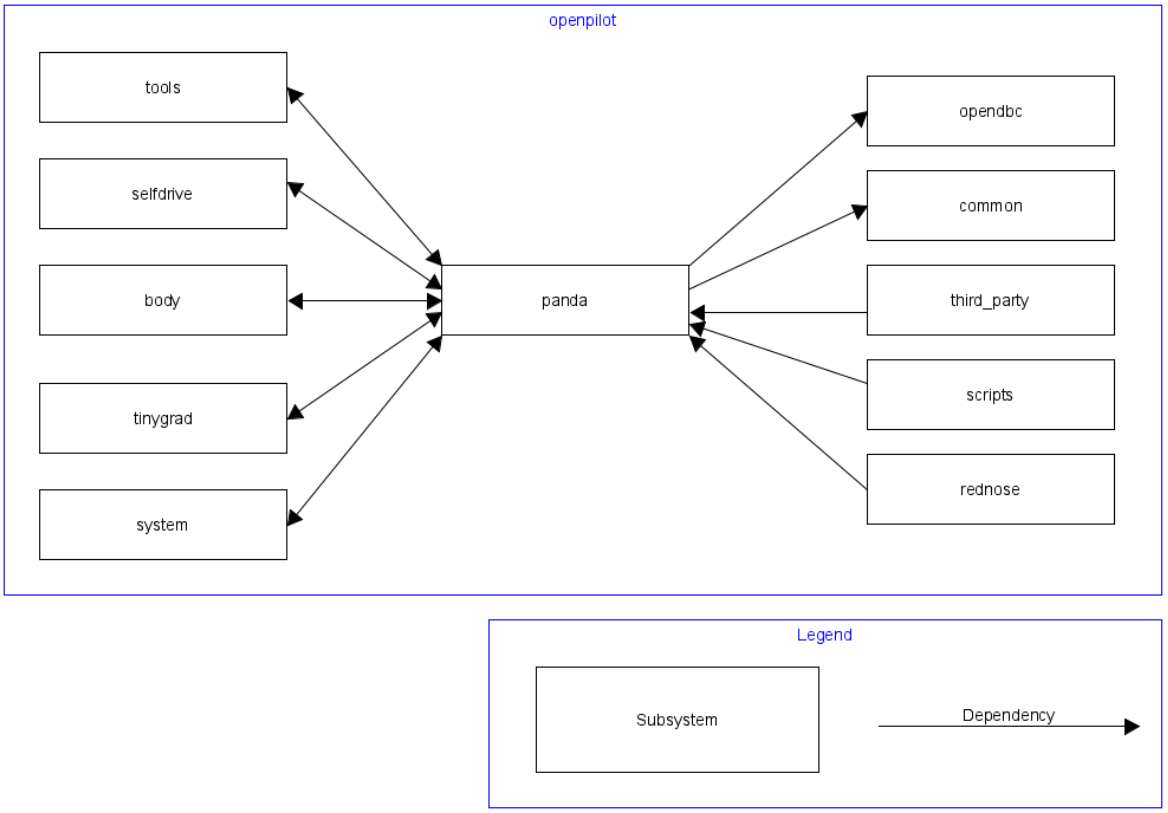

Figure 9: Panda's Extracted Dependencies.

Comparison of Panda Subsystem

Reflexion Model Analysis:

Conceptual vs. Concrete Model: The conceptual model of the panda Subsystem was initially thought to be driven by an event-driven (implicit invocation) architecture with minimal dependencies. However, upon closer inspection of the concrete model, we discovered a more intricate dependency structure. This discrepancy highlights the need for a reflexion model to bridge the gap between the two models.

Venn Diagram Comparison: A Venn diagram comparison would illustrate the overlaps and differences between the conceptual and concrete models. It would show the convergences in components like opendbc, panda, and the CAN bus, which are consistent across both models. Conversely, it would also highlight the divergences such as the unexpected dependencies found in the concrete model as well as the absences which are present in the conceptual model but missing from the concrete model. They will be further discussed in component analysis below.

Event-Driven Architecture: While the conceptual model claims an event-driven architecture, the concrete model reveals that panda has many more dependencies. For instance, other subsystems might use cereal to translate communications with panda, even though panda never directly interfaces with cereal.

Dependency Analysis: In the concrete model, the number of dependencies is higher than anticipated (6 more dependencies). A detailed count of these dependencies is crucial for a comprehensive reflexion model. This count will inform how we can adjust our conceptual model to better match the concrete reality.

Revised Conceptual Model: To align the conceptual model with the concrete findings, we should incorporate the additional dependencies into the conceptual framework. This includes recognizing the role of cereal and the various subsystems that indirectly interact with panda.

Terminology: The term "reflexion model" is central to this analysis, as it encapsulates the process of reflecting on the conceptual model and adjusting it to mirror the concrete model accurately.

Components Analysis:
Divergences: Components like tinygrad, selfdrive, tools, common, body, and system show a more complex system than the conceptual model suggested. These can be categorized into Decision-Making, Utility, and Hardware Communication.
Convergences: opendbc, panda, and the CAN bus are consistent between the models, serving as the communication framework.
Absences: The absence of cereal in the conceptual model is a significant oversight, as it plays a vital role in communication translation.
Necessity of Actuators and Sensors: The Collection of Actuators and Collection of Sensors were not removed in the conceptual model. They are essential for understanding the system's communication, even though they do not directly interface with panda.

By addressing these points, we can refine the reflexion model to provide a more accurate representation of the panda Subsystem.

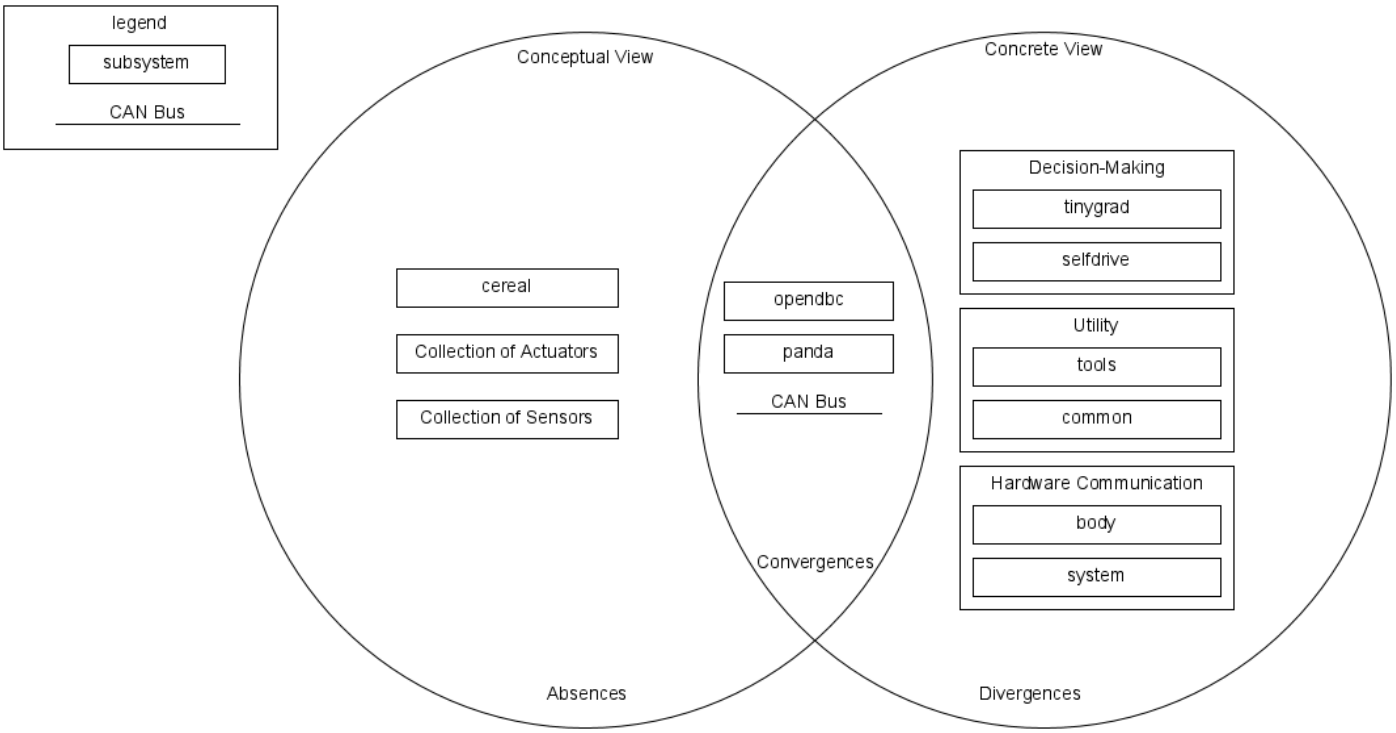

Figure 10: Venn Diagram of Conceptual vs Concrete Panda Architecture

The figure below, which focuses on the dependencies and communications between the subsystems, is instrumental in illustrating the actual interactions within the system. It is this detailed mapping of relationships that explains the differences noted in the reflexion analysis. By accurately depicting how the subsystems are interlinked, and how they communicate and depend on one another, the diagram provides a visual justification for the changes made from the conceptual to the concrete model. Specifically, cereal is communicating through the decision-making, hardware communication, and utility groups and not directly connected with panda. The collection of actuators and sensors are also not directly connected to panda and they communicate through the CAN bus.

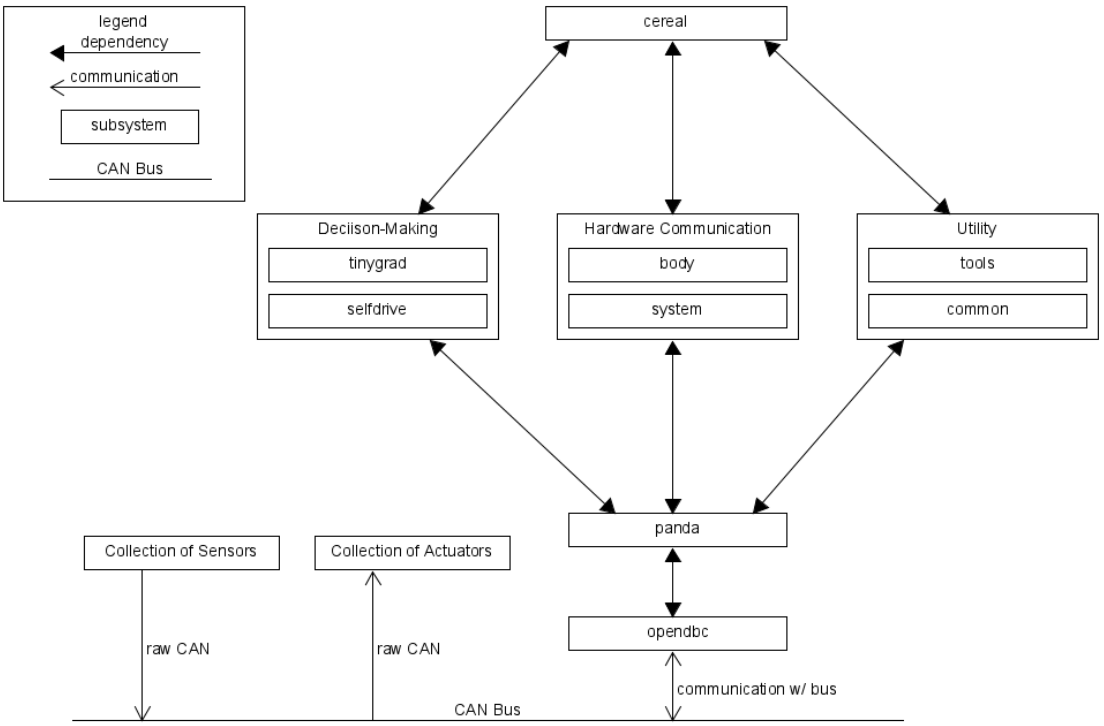

Figure 11: The Revised Architecture of panda with Dependency and Bus Communication

Concurrency

In report 1, we highlighted the presence of concurrency in our conceptual architecture analysis. This assertion was based on the multiple inputs the openpilot system receives from various different sources, including cameras and sensors within the vehicle. These inputs are utilized together to generate different outputs. Additionally, concurrency was identified in the conceptual architecture through the likelihood that the decision-making model utilizes the different inputs to distribute data to software components. These components may leverage overlapping input from the same sensors, further portraying the concurrency of the system.

In report 2, we highlighted the presence of implicit invocation mechanisms such as publish/subscribe and event-driven bus architectures in the concrete architecture. These mechanisms facilitate concurrency among various components. For instance, panda demonstrates this concurrency by processing serial data from a USB port in real-time while simultaneously performing tasks like encryption and CAN translation. Moreover, panda

exhibited concurrency by transmitting data serially through ports, seamlessly integrating with the ongoing activities of other processes.

In comparing our findings from report 1 and report 2, it's clear that while our conceptual analysis correctly identified the presence of concurrency in the openpilot system, it only provided a partial understanding of its extent and implementation. In report 1, we correctly identified the utilization of multiple inputs from various sources, indicating concurrency in the conceptual architecture. However, our analysis in report 1 failed to fully grasp the comprehensive nature of concurrency within the system. Report 2 revealed a more detailed picture of concurrency in the concrete architecture. Report 2 highlighted the use of implicit invocation mechanisms such as publish/subscribe and event-driven bus architectures. These mechanisms enable concurrency among various components of the system. The example of panada processing serial data while simultaneously executing other activities showcases the more detailed nature of concurrency in the concrete architecture.

Moving forward, to align our conceptual architecture analysis with the concrete architecture analysis, we can enhance our understanding by performing a more comprehensive exploration of the systems architecture. Furthermore, we should focus on identifying and analyzing mechanisms such as implicit invocation present in the conceptual architecture to better predict the extent of concurrency within the system. By making these changes, we can more accurately analyze the concurrency in the conceptual architecture of the system.

After analyzing issues within openpilot's github repository, it became clear that there were moments of disagreements or differing viewpoints among the team in terms of how to implement changes/solve problems in the system. These disagreements seem to often arise due to differences in things such as technical approaches, time constraints and project priorities.

Lessons Learned

One key lesson learned from completing Assignment 3 includes the beneficial use of reflexion models. This assignment allowed for a deeper understanding of how to compare one's original conceptual view with the actual implementation (concrete view) of a system. The use of reflexion models allowed us to investigate the original gaps in our conceptual predictions, and determine where the absences and divergences fall. We also learned about how this view allows us to focus on the divergences found, which we initially missed in the conceptual view. Further exploration of these divergences helps us to understand sections of the code that we had originally missed, and which steps are required to revise our conceptual view.

Another lesson learned from this assignment is that documentation and code base often mismatch and the documentation is outdated. In Assignment 1 the conceptual view was constructed based on documentation, including the openpilot website and ReadMe files on their GitHub. However, once we examined the source code and constructed the reflexion model, we see there are numerous dependencies that we were completely unaware of, including the components for decision-making, utility, and hardware communication. This is an important lesson learned as Software Engineers, to gain exposure to the difference in the apparent architecture, from a user perspective, to the concrete architecture, from a developer perspective.

This entire process of EECS4314 Assignments 1, 2, and 3 allowed for our group to gain incredible insight into the processes involved in extracting information from documentation and source code. We've been able to directly apply the contents from class to a real world system, and further our understanding of the complex architecture of large systems. This has allowed us to prepare for graduating from our undergraduate program, and applying the concepts from classroom lectures to real world scenarios. This has been an incredible opportunity regarding the large level systems we can expect to work on within industry.

If we were to repeat this process, one thing that we'd do differently is that we'd consider reference architecture as well when forming our prediction about the conceptual architecture. Although it may be difficult to find a close reference architecture to use, even referencing other large systems could have helped with the conceptual architecture being more complete. A specific example of this is in the "utility" component was a divergence, as it was left out of our conceptual view. However, it is quite common for large systems to have a central utility component for common methods.

Conclusion

Our findings highlighted several discrepancies between the conceptual architecture and the implemented concrete architecture, particularly in the dependencies and presence of concurrency throughout the system. This was because only online documentation was used to design the conceptual architecture rather than source code for the conceptual model.

The conceptual models, while useful for initial framing and understanding, could not fully anticipate the developer decisions that shaped the final architecture. Our analysis revealed that the openpilot system, especially within the panda subsystem, operates with a level of concurrency and dependence far beyond our initial predictions. Openpilot specifically uses implicit invocation mechanisms such as public/subscribe and event-driven bus architectures which were underlooked in our conceptual architecture. This is driven by the system's need for real-time responsiveness and robust data handling.

The high-level conceptual model, which was described using MAPE-K and Client-Server, which also did not fully align with the concrete implementation. Furthermore, we learned that panda had no direct communication with the CAN bus, sensors or actuators and that this communication was instead primarily managed through opendbc. The critical lesson from this analysis emphasizes the necessity of using reflexion models and source code review. These techniques reveal the gap between documentation-based assumptions and the actual system design.

References

[1] "Delft students on software architecture," Delft Students on Software Architecture - DESOSA 2020, https://desosa.nl/ (accessed April 1, 2024).

[2] "How openpilot works in 2021," comma.ai blog, https://blog.comma.ai/openpilot-in-2021/ (accessed April 1, 2024).

[3] Lightweight self-adaptive configuration using machine ..., https://www.cs.ubc.ca/~rtholmes/papers/cascon_2021_araujo.pdf (accessed April 1, 2024).