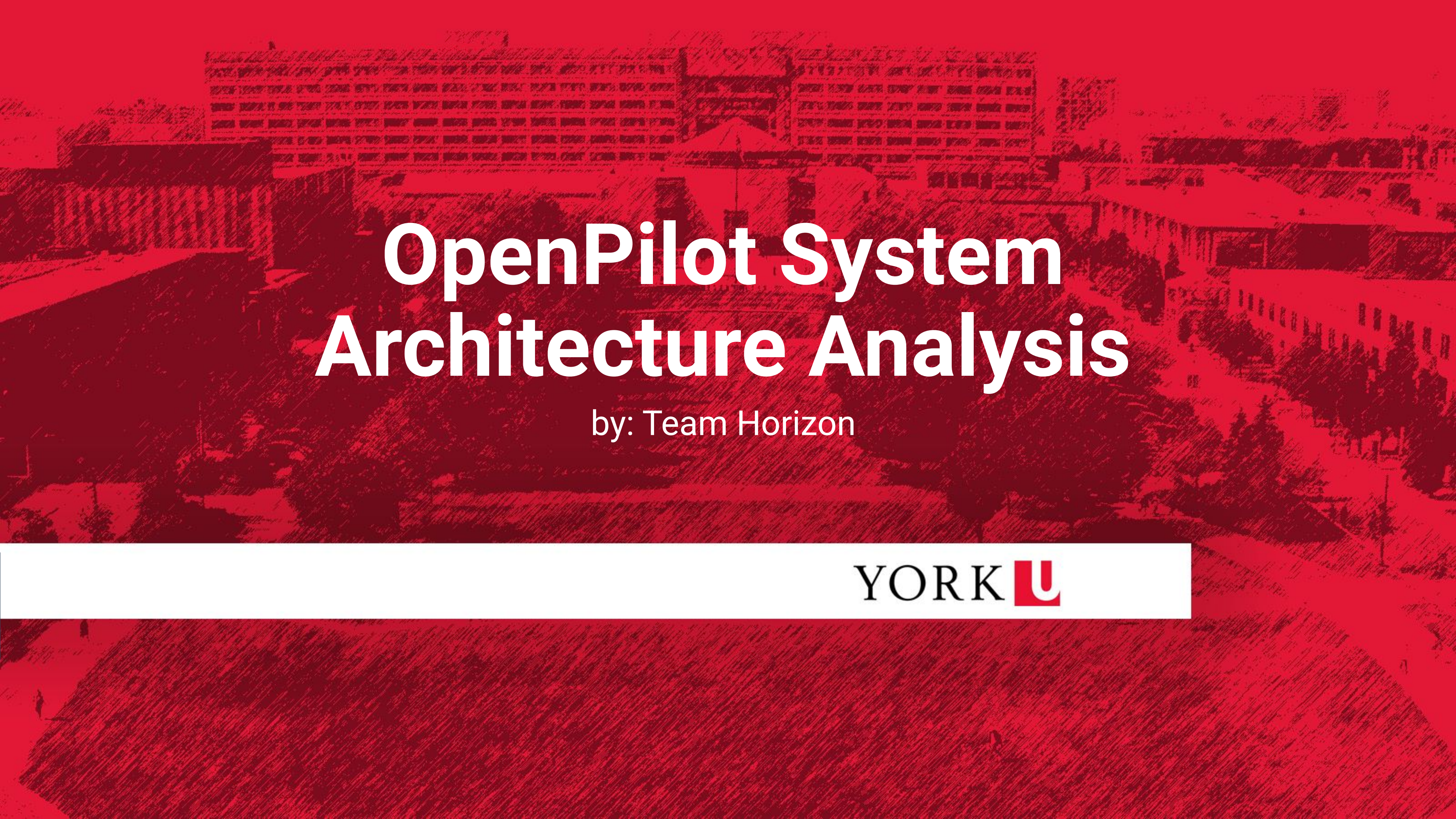


OpenPilot System Architecture Analysis

by: Team Horizon

YORK U

Agenda

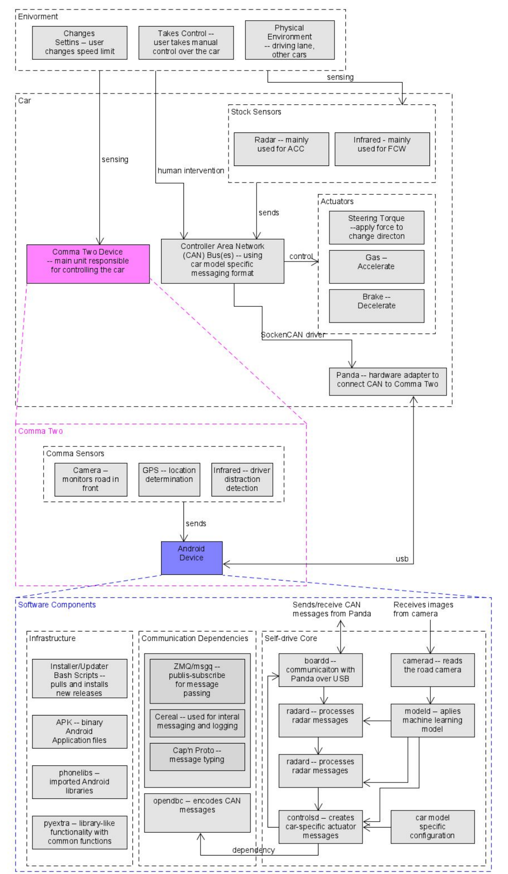
1. Analysis Process
2. Top Level of Subsystems and Interactions
3. Comparison of System Architecture
4. Top Level of Panda Subsystem
5. Comparison of Panda Subsystem
6. Concurrency
7. Lessons Learned
8. Conclusion

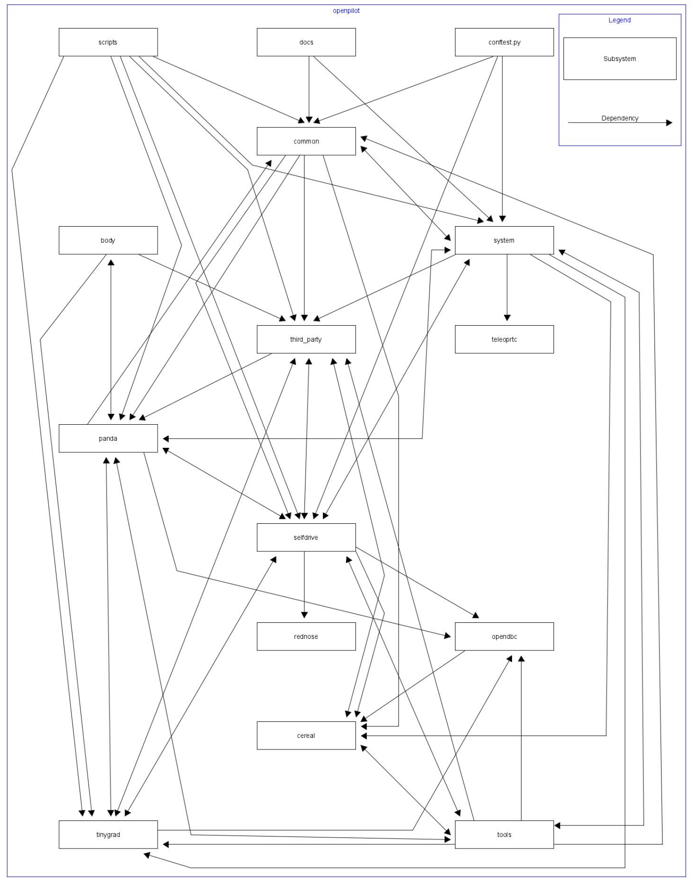
Top Level of Subsystems and Interactions

- Report 1 - Conceptual
  - **Architectural Styles:** Layered, Implicit Invocation, Process Control, Client-Server, Pipe and Filter, Repository
  - **Subsystems:** Sensors, actuators, neural networks, localization, controls, logging, hardware
  - **Key Components:** panda, openDBC, board, cereal
  - **Functionality:** Data and communication flow, concurrency

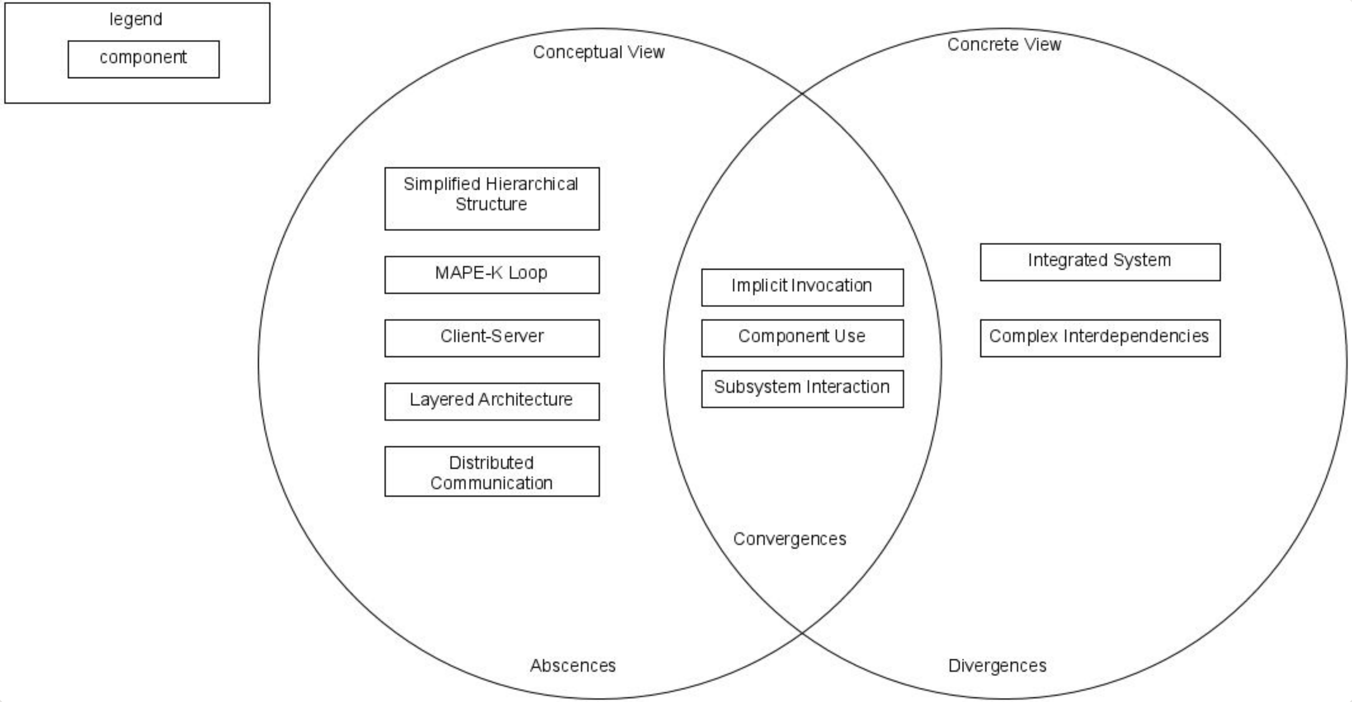
- Report 2 - Concrete
  - **Focus on panda:** using Understand and its role in Openpilot
  - **Dependencies** on panda
  - **Data and communication** flows
  - Found **implicit invocation**
  - **No** design patterns

Top Level of Subsystems and Interactions


Enivorment
Changes Settins -- user changes speed limit
Takes Control -- user takes manual control over the car
Physical Environment -- driving lane, other cars
sensing
Car
sensing
human intervention
Stock Sensors
Radar -- mainly used for ACC
Infrared - mainly used for FCW
sends
Actuators
Steering Torque --apply force to change directon
Gas -- Accelerate
Brake -- Decelerate
control
Comma Two Device -- main unit responsible for controlling the car
Controller Area Network (CAN) Bus(es) -- using car model specific messaging format
SockenCAN driver
Panda -- hardware adapter to connect CAN to Comma Two
Comma Two
Comma Sensors
Camera -- monitors road in front
GPS -- location determination
Infrared -- driver distraction detection
sends
Android Device
usb
Software Components
Sends/receive CAN messages from Panda
Receives images from camera
Infrastructure
Installer/Updater Bash Scripts -- pulls and installs new releases
APK -- binary Android Application files
phonelibs -- imported Android libraries
pyextra -- library-like functionality with common functions
Communication Dependencies
ZMQ/msgq -- publis-subscribe for message passing
Cereal -- used for interal messaging and logging
Cap'n Proto -- message typing
opendbc -- encodes CAN messages
Self-drive Core
boardd -- communicaiton with Panda over USB
camerad -- reads the road camera
radard -- processes radar messages
modeld -- aplies machine learning model
radard -- processes radar messages
controlsd -- creates car-specific actuator messages
car model specific configuration
dependency



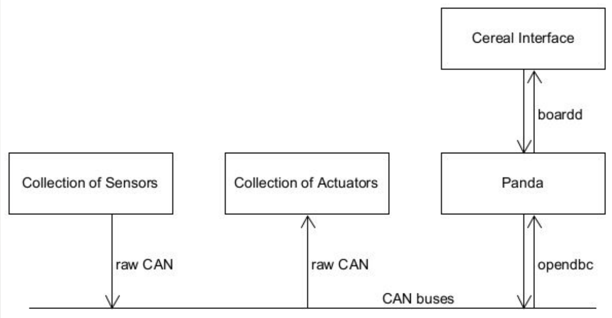
Comparison of System Architecture



Top Level of Panda Subsystem

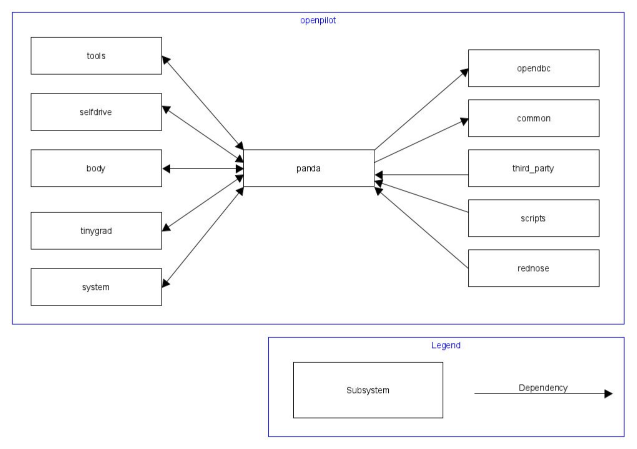
Conceptual

- Dependencies
  - boardd, opendbc, cereal
- Communication
  - CAN buses, collection of sensors, collection of actuators

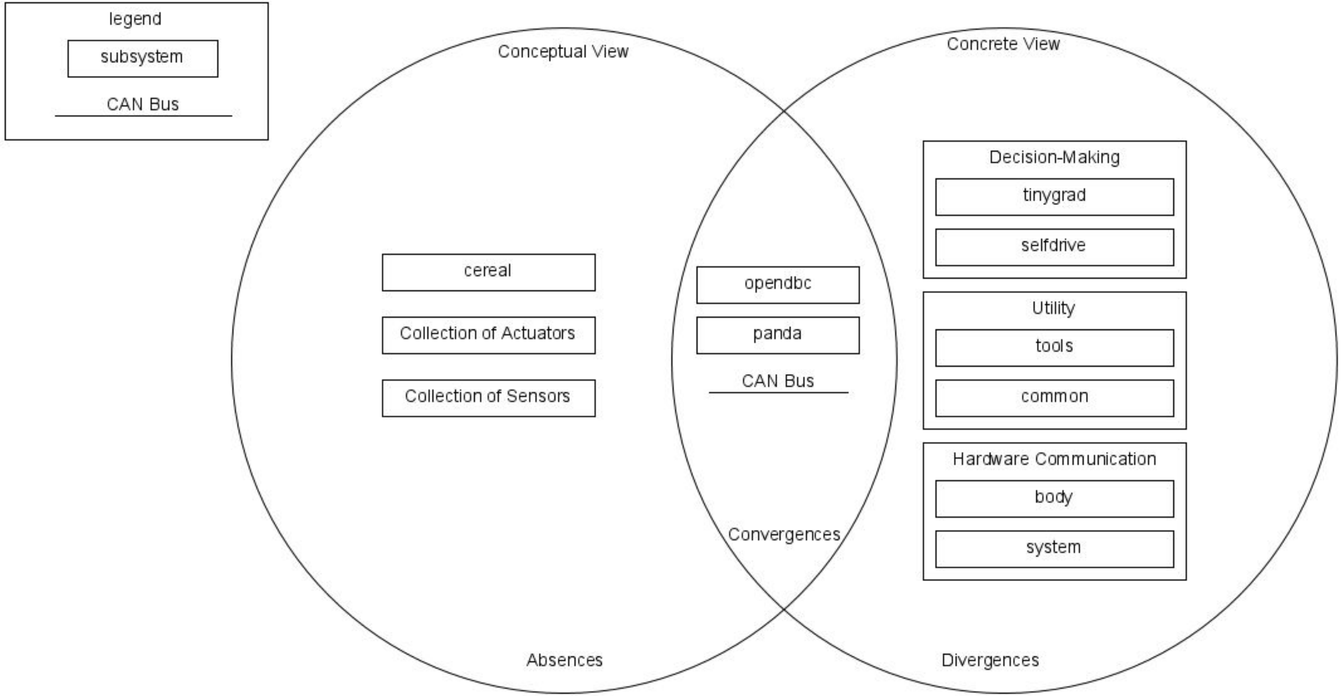


Concrete

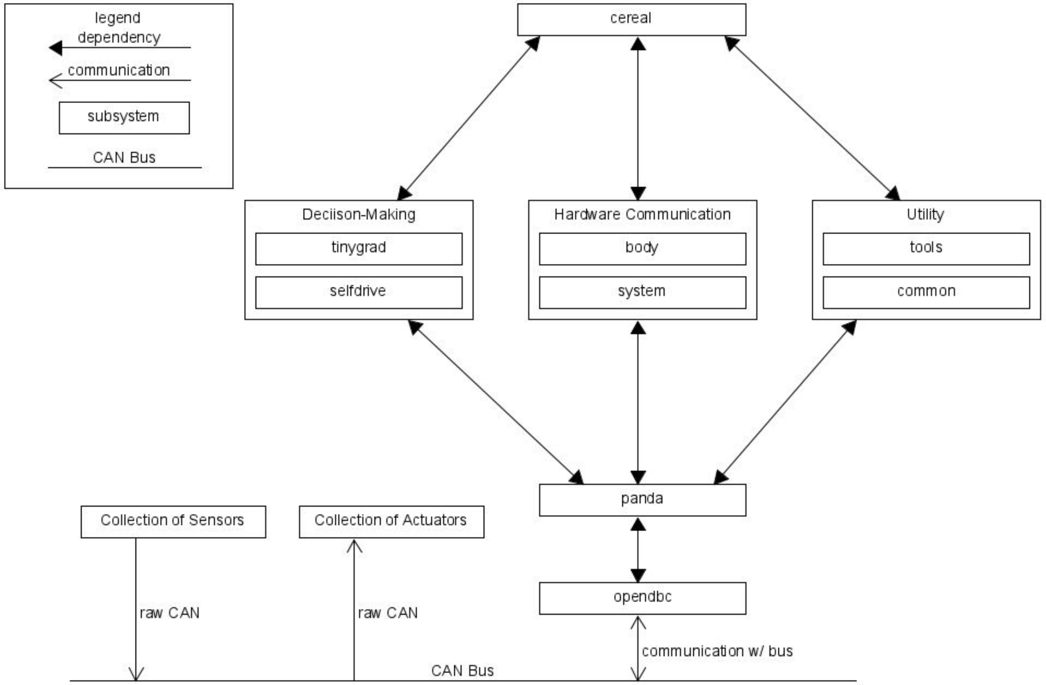
- Dependencies
  - tools, selfdrive, body, tinygrad, system, opendbc, common



Panda Comparison



Panda Remade



Concurrency

- Report 1 highlighted concurrency in the conceptual architecture
  - Concurrency Identified based on multiple inputs (e.g., cameras, sensors)
- Report 2 emphasized concurrency in concrete architecture
  - Implicit invocation mechanisms (e.g., pub/sub, event driven bus architectures) enabling concurrency
- Comparison
  - Report 1 correctly identified concurrency but lacked comprehensive understanding of its extent
  - Report 2 provided a detailed picture of concurrency, through the use of implicit invocation mechanisms
- Solutions
  - Perform a more comprehensive exploration of the conceptual system architecture
  - Focus on identifying and analyzing mechanisms such as implicit invocation to better predict concurrency

Team Issues

- Team Issues
  - Issues within Openpilots Github repo showed disagreements or differentiating viewpoints among the team.
  - Disagreements often due to differences in technical approaches, time constraints, and project priorities

Lessons Learned

- Beneficial Use of Reflexion Models
    - Reflexion models provide a clear method of comparison when comparing conceptual and concrete software architectures.
    - Clear & organized result in terms of absences, convergences, and divergences.
    - Divergences shows which components we failed to consider in the conceptual view.
- Gained experience and understanding of how documentation and source code can mismatch.
    - From a user/feature perspective, we expected the system to be structured a certain way. This was different from implementation.
- Something we'd do differently - Reference other system architecture to create the conceptual model in Assignment 1.

Conclusion

- Discrepancies in Architecture: Gaps in dependencies and concurrency from documentation reliance.
- Complexity Underestimated: Missed prediction of concurrency and invocation mechanisms.
- Expectations vs. Reality: Initial models (MAPE-K, Client-Server) misaligned with implementation.
- Reflexion and Code Review: Essential for bridging theory and actual system design.

Thanks for your attention!

Any questions/ suggestions/ concerns?