SE 339 - Assignment 3
UML Models

10/21/19

Authors:

Stamatios Morellas (*morellas@iastate.edu*)
Richard Smith (*gsmith7@iastate.edu*)

Understanding the System

1. Identify inputs and outputs
    a. The system reads data from the vehicle including, usage statistics, driver information, and vehicle information for a particular fleet *(Input)*
    b. The system transmits data from the vehicle to the server *(Output)*
2. How does the system collect data?
    a. The system collects data through the vehicle's On-Board Diagnostics II (OBD-II) port
3. How does the system store the data?
    a. The system shall record vehicle data into a database
    b. The Fleet Data Collector creates and stores a log in a .txt format each time the service is run
4. How does the system manipulate the data?
    a. The system shall transform the raw in-vehicle data to data that humans can understand on the server
    b. The system shall display a map with the locations of all the vehicles in the fleet
    c. The system shall display live data, such as speed, engine temperature of a given vehicle
    d. The system shall allow managers to register or remove the vehicles that belong to a particular fleet
    e. The system shall allow the managers to customize the data being displayed to them
5. How do the components communicate with each other?
    a. The vehicle microservice communicates directly with the database
    b. The driver-microservice communicates with the database by also communicating with the vehicle-microservice
    c. The usage-microservice communicates with the database by also communicating with the driver-microservice AND the vehicle-microservice

Use Case Diagram

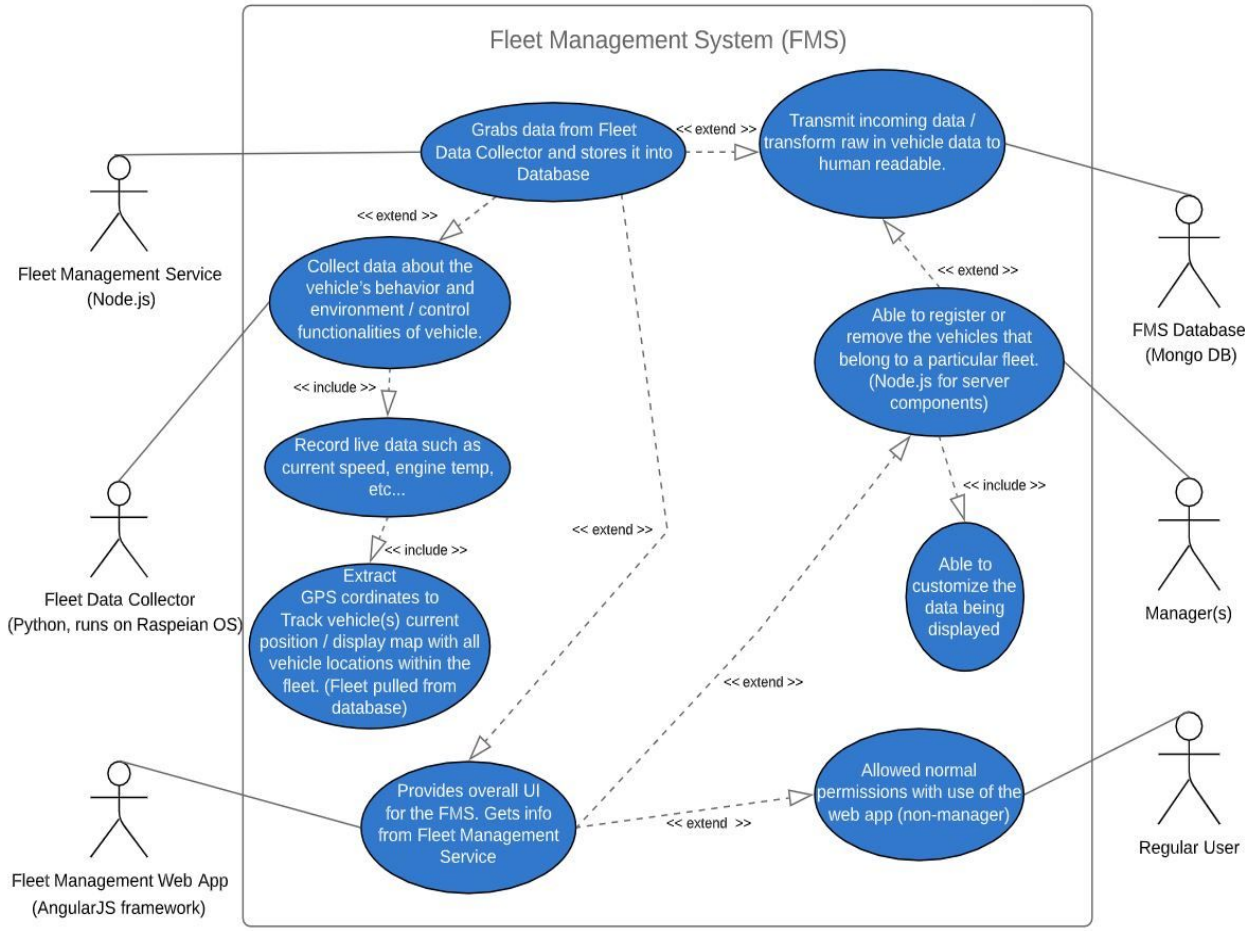

Figure 1.1: Use case diagram listed with actors and system functionalities

As you can see, there are 6 Actors for this Use Case Diagram. The Fleet Management Service, Fleet Data Collector, Fleet Management Web App, Fleet Management Service Database, Managers, and Regular Users. The Fleet Management Service runs off of Node.js and grabs data from the Fleet Data Collector. Then Fleet Management System sends info to the Fleet Management Web App and the database, and from there the Regular User / Managers can interact with the Web App and database as shown above.

Actor	**Description**
Fleet Management Service	Grabs data from Fleet Data Collector and sends to database and webapp
Fleet Data Collector	Collects data from vehicle and initiates devices and services for extracting data form the in-vehicle network
Fleet Management Web App	Visualizes the fleet management data it gets from the Fleet Management Service to the user
FMS Database	Stores all data that Fleet Management Service Extracts from Fleet Data Collector
Managers	Super user of the web app
Regular User	Normal user of the web app

Component Diagram

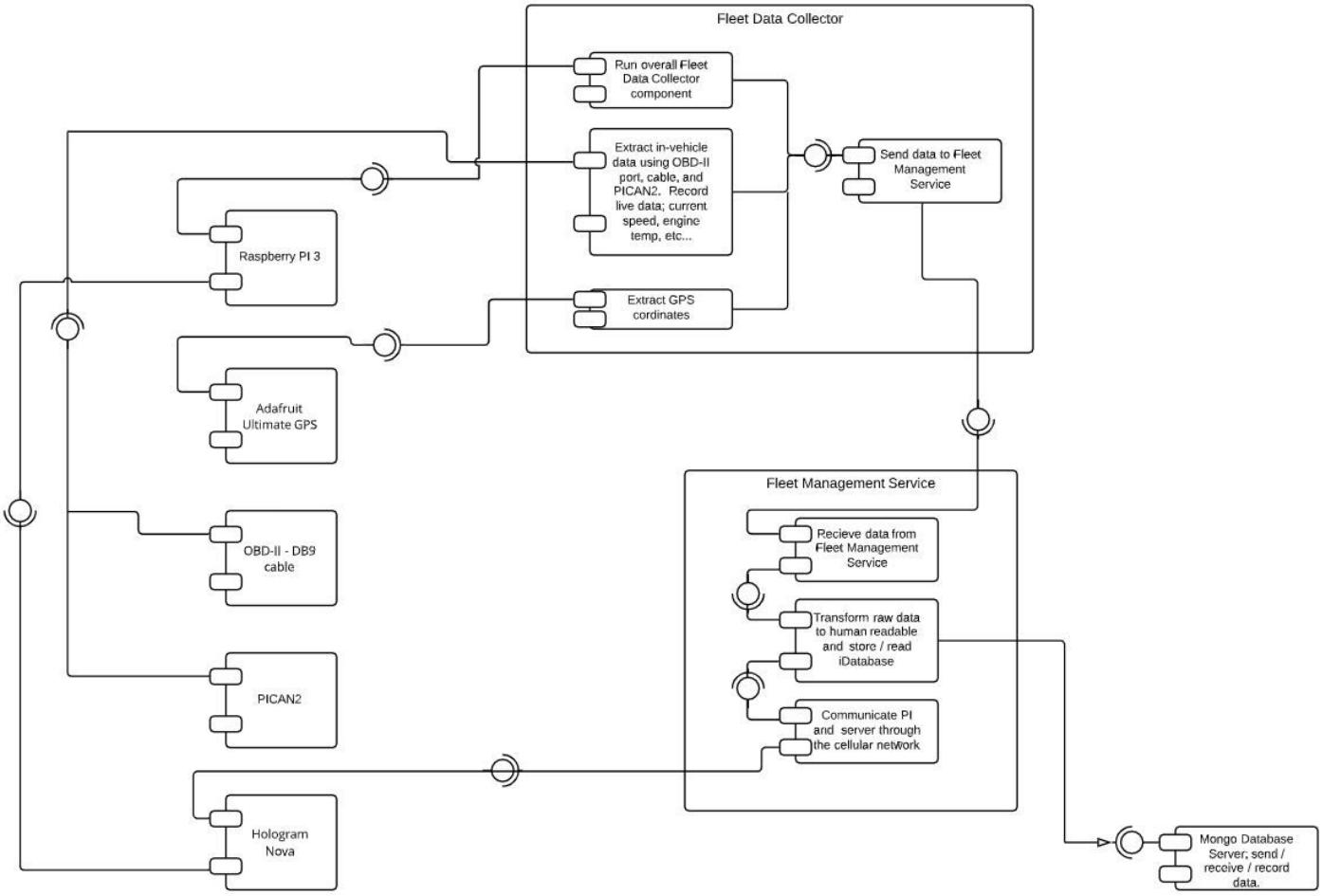

Figure 1.2: Component diagram listed with all the components that make up the Fleet Management System

Raspberry PI 3	Runs the Fleet Data Collector Component
Adafruit Ultimate GPS	Gets the current position
OBD-II - DB9 cable	Connects the PICAN 2 to the vehicle OBD-II
PICAN2	Communicates with vehicle CAN-bus
Hologram Nova	Communicates the PI and the server through the cellular network

Communication Diagram

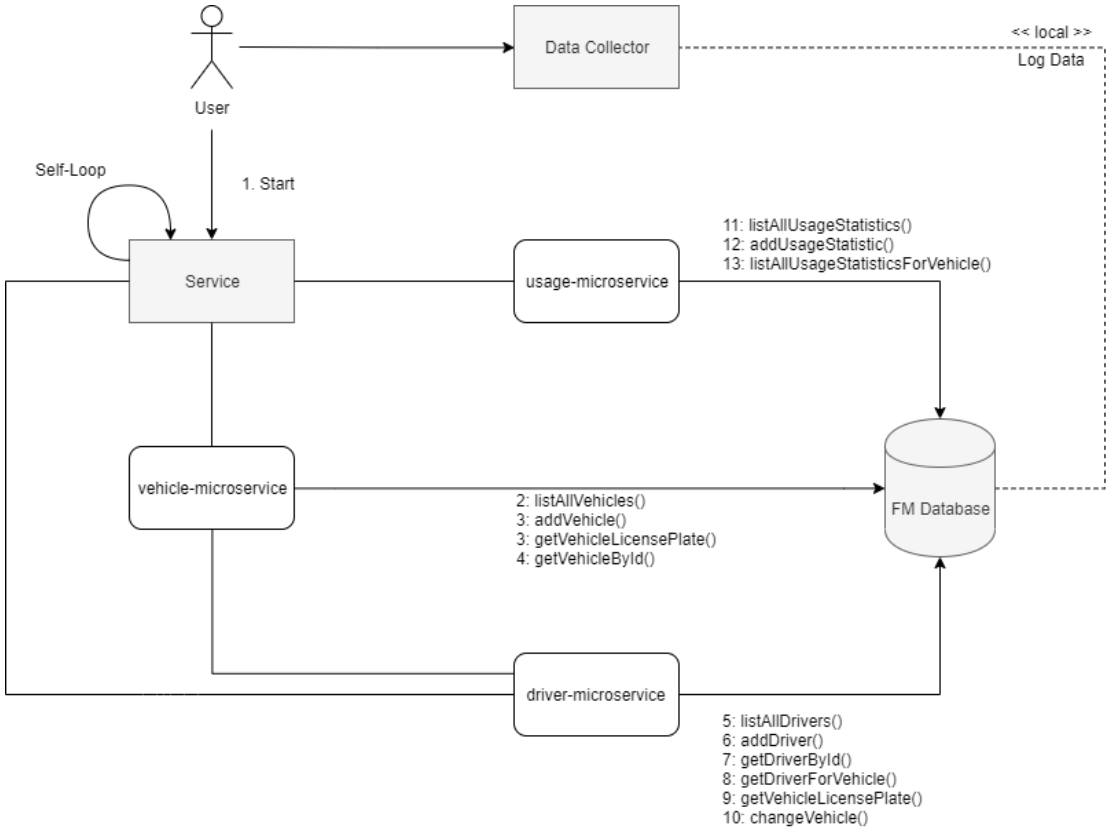

Figure 1.3: Communication diagram explaining how the various components in the Fleet Management System interact with each other

The communication diagram shows how different elements in the Fleet Management System interact with each other in order to satisfy the various use cases. The user first must activate the data collector as well as the service itself. When testing the service, we executed the code in IntelliJ Java IDE, but in an ideal situation, the service would be started from a more user friendly interface such as a web application. The data collector component logs the data locally from the database server during its uptime usage. It stores the live data of the vehicle and sends it to the database so that it may later be visualized in a web interface. Meanwhile, the service is able to pull the driver, vehicle, and usage information from the database in addition to modifying the information for these three categories. The service will run indefinitely until shut down by the user.
(Can't get rid of the yellow highlighting for some reason)

Reflection

Both team members had previous experience working with UML diagrams (classes and recitation), so it wasn't too difficult to understand the purpose and function. UML diagrams can be extremely useful in identifying and visualizing how components in a software architecture interact with each other, which helps in determining the overall operation of a system. By breaking down the structure of a system, UML diagrams are able to partition large systems into smaller pieces in order to make it easier to understand how the larger system works as a whole by first understanding the more individual parts. Our team found it easier to create UML diagrams from software that already has a working architecture. This is due to the fact that we could test the system ourselves and derive which components are related to each other and how each element integrates into the overall system. Alternatively, if the software has not been built yet, creating a UML diagram can be a very essential starting point that would make the development process much easier. It is worth acknowledging that if this is done, the actual software that is produced may sometimes differ slightly from the expected outcome that was initially planned for in the UML diagram. This would mean that although the diagrams wouldn't use a different format, they would only differ based on their representations of the software architecture.