RE-Vehicle Aggregator App

System Design Document

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Prepared By | Review By | Version No |
| 1 | Ashwath Kumar K | Hareesha/Aparna | 1.0.0 |
|  |  |  |  |

Table of Contents

[1.System Overview 2](#_Toc150243152)

[2.1 High Level Design – Block Diagram 3](#_Toc150243153)

[2.2.1 Hardware Communication Layer (H/W Communication Layer) 4](#_Toc150243154)

[2.2.2 Communication Driver Layer 4](#_Toc150243155)

[2.2.3 Data I/O Interface 4](#_Toc150243156)

[2.2.4 Data Preserve Layer 4](#_Toc150243157)

[2.2.5 Signal Conversion Layer 5](#_Toc150243158)

[2.2.5 Addition Service Layer 5](#_Toc150243159)

[2.2.5 Common Data Layer 5](#_Toc150243160)

[3. Classes and its Relation 5](#_Toc150243161)

[4. Storage Strategy 5](#_Toc150243162)

[4.1 Available storage strategy 6](#_Toc150243163)

[4.1.1 Buffering 6](#_Toc150243164)

[4.1.2 Streaming and Event-Driven Architectures 7](#_Toc150243165)

[4.1.3 Caching 7](#_Toc150243166)

[4.2.4 Load Balancing 7](#_Toc150243167)

[4.2.5 Asynchronous Processing 7](#_Toc150243168)

[5 . Implementation 8](#_Toc150243169)

[6 . Data Matrix 8](#_Toc150243170)

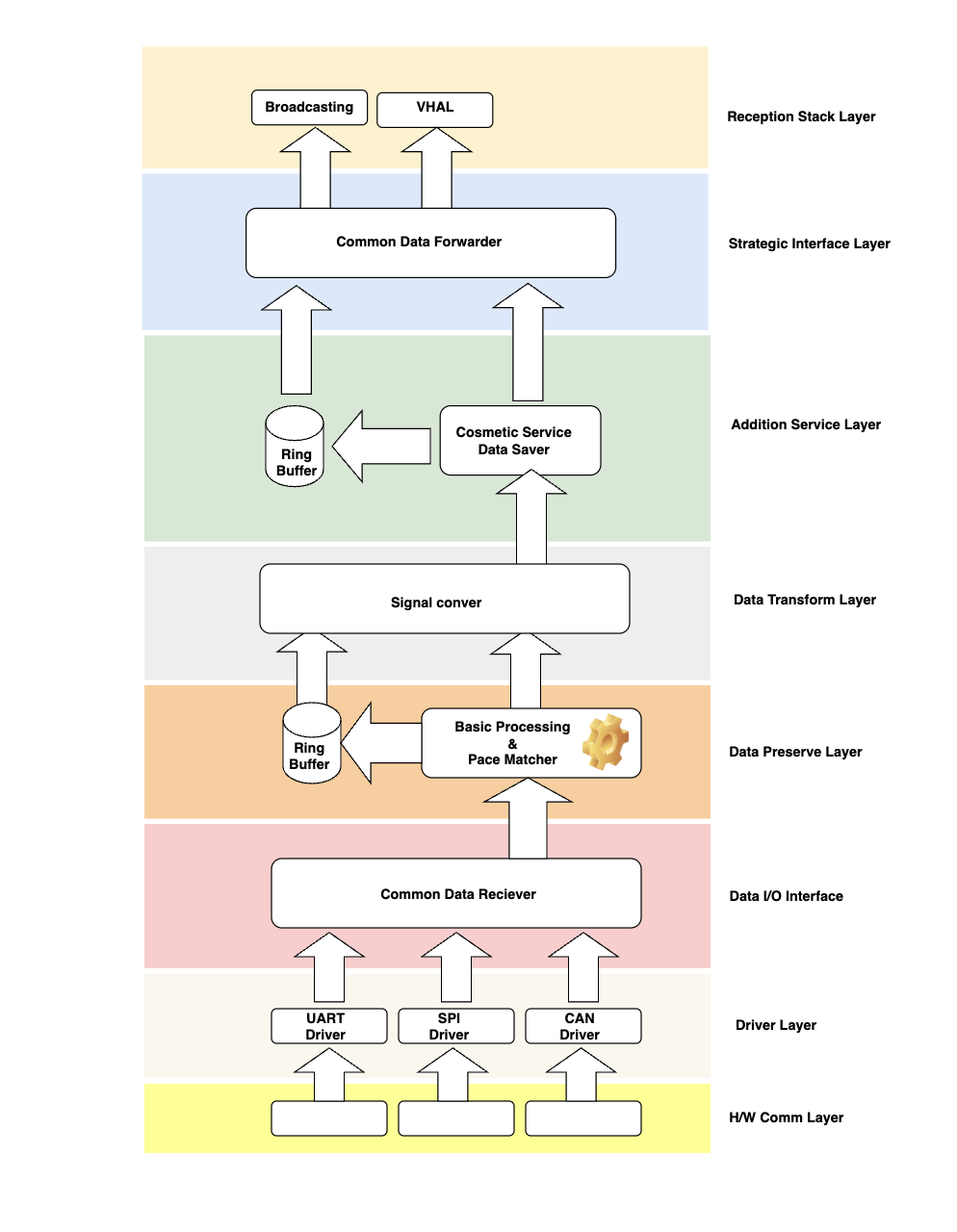
[7. Test Plan 8](#_Toc150243171)

[8.Bibliography 8](#_Toc150243172)

# 1.System Overview

Data aggregator is a service in AOSP for getting the data from lower level, polishing it and forwarding or exposing interface to other services that require signal or CAN packets. In the automotive industry there is a different flavor of standard. automotive software AOSP package is available, specifically for four-wheeler. But in case of two-wheeler there is no standard software packet available which contains all the basic necessary which is suitable for two wheeler. Some companies like Qualcomm are building a miniature version of software that behaves like automotive and mimicking it but is not the same. That is why all the companies involved in two-wheeler are generally taking bare AOSP from microcontroller/microprocessor providers that aid the WIFI, Bluetooth and other necessities hardware.

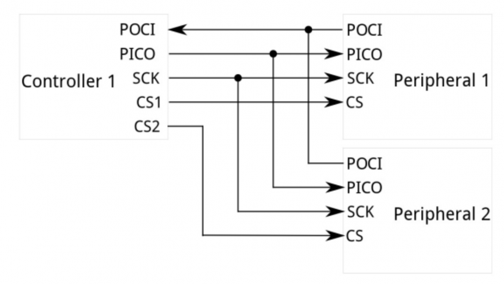
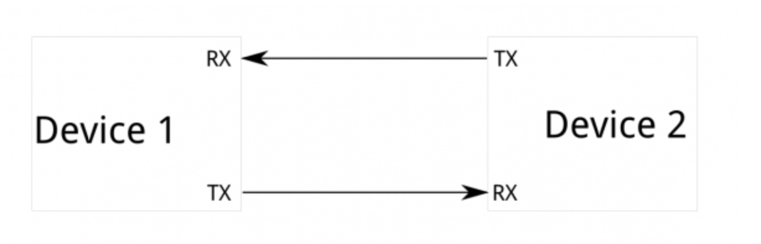
# 2.1 High Level Design – Block Diagram



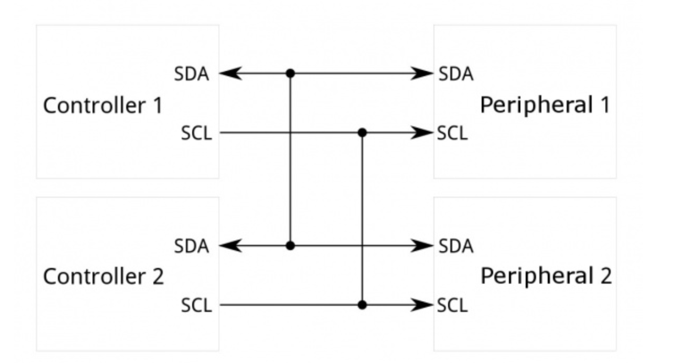
Above pic depicts the HLD of Data Aggregator. Each layer of the above HLD has its own responsibilities and the layer structure of the data aggregator provides the facilities to change anything/adapt anything in the respective layer without touching or disturbing the other layer.

## 2.2.1 Hardware Communication Layer (H/W Communication Layer)

This layer is responsible for the hardwire connection between the two devices that want to communicate with each other. Examples are UART, I2C, SPI



UART SPI



I2C

## 2.2.2 Communication Driver Layer

Communication driver layer is responsible for picking the signal from the communication hardware layer, understanding it and making it in digital format and providing the interface to other layers to read it and write it. Generally, there is one to one mapping between the Communication driver layer and communication hardware layer.

## 2.2.3 Data I/O Interface

As we are building the DA so for this particular service data source might be different, but it should be abstracted in such a way that for all the layers which are responsible for consuming the data pretend the source is single. The work of Data I/O interface is to get the data from different sources but provide the single interface for all above layers which are consumers.

## 2.2.4 Data Preserve Layer

Idea of data preserving layer is to buffer if data is too fast from the sender either from upper layer or lower so by introducing this layer, we will get the data losses issue from consumer issue due to pace difference.

## 2.2.5 Signal Conversion Layer

Objective of the Signal conversion layer is to dedicatedly do conversion/polishing/filtering etc of the signal. Making it separate leads to several benefits like strategically we can plug it and plug out the different conversion methodology/algorithm.

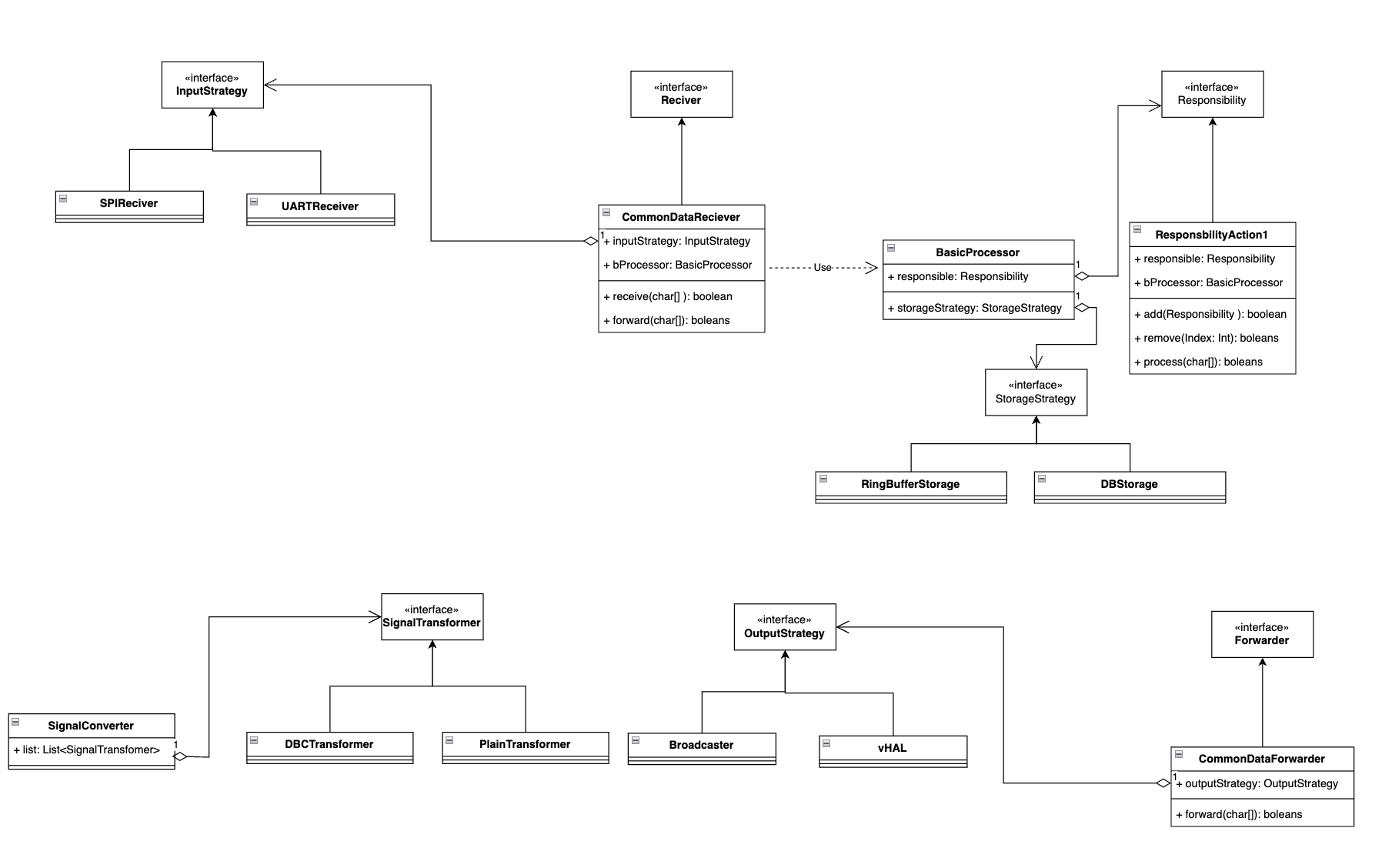
## 2.2.5 Addition Service Layer

In addition to the service layer, we can add the different subservice like security, topic-based protection, signal wise buffering, handshaking, etc.

## 2.2.5 Common Data Layer

Common data provider layer acts as the source for the outer layer, It will expose the interface for different strategies for providing the data example, Intent based broadcasting, AIDL based interface etc.

# 3. Classes and its Relation

****

The above block diagram represents classes and relations among them.

# 4. Storage Strategy

A diagram of a computer program

Description automatically generated with medium confidence

In the data preserve layer and addition service provider layer, we have mentioned the buffer to hold the data in case there is a difference between speed of consumer and sender. Selection of buffer storage is totally dependent upon your requirement and design, to choose the storage type a few things we need to consider like - frequency of data, size of data and type of data. Generally following option are available for buffer storage.

1. RAM buffered space.
2. Database
3. Cached based buffer.
4. Cached based DB.

All the storage strategies that mentioned above are performed optimally based on the use cases. Let application wants driver/user-based document to forward to BMS than index based database is performed optimally similarly if upper application or lower application want sending the command or some critical signal to respective h/w then it has to process before trigger the hardware so we have to buffer to avoid the loss in this case we can choose cached based buffer or ram based buffer.

## 4.1 Available storage strategy

The choice of the best data storage strategy for managing the speed of data consumers and data senders depends on the specific requirements of application, especially in terms of data volume, data velocity, and latency constraints. Here are a few strategies to consider:

## 4.1.1 Buffering

We can use buffers to temporarily store data when there is a mismatch in speed between producers and consumers. This is particularly useful in scenarios where the data rate fluctuates or when consumers need time to process the data.

Buffer size and management play a crucial role in controlling latency. If the buffer becomes too large, it can introduce excessive latency; if it's too small, data may be lost.

Implementing circular buffers or bounded queues can help in managing buffering efficiently.

Since in vehicle aggregator Data going to be huge, we really need to think about buffering carefully.

## 4.1.2 Streaming and Event-Driven Architectures

In scenarios where low latency is a requirement, a streaming architecture might be more appropriate. Data is pushed to consumers as soon as it's produced. Event-driven architectures, using tools like Apache Kafka or RabbitMQ, can be useful for handling high-velocity data streams. Real-time data processing systems can benefit from these architectures to ensure timely delivery.

## 4.1.3 Caching

Caching can be used for both data producers and consumers. Producers can cache data to provide it more quickly to consumers, and consumers can cache data they've already processed for faster retrieval. Caches can help reduce the load on data stores and databases, speeding up access to frequently requested data.

## 4.2.4 Load Balancing

If you have multiple data producers and consumers, load balancing can help distribute the data processing load evenly. Load balancing strategies ensure that data is evenly distributed among consumers, preventing bottlenecks.

## 4.2.5 Asynchronous Processing

For consumers that don't require real-time data, asynchronous processing can be beneficial. Data is collected and processed in the background, allowing consumers to retrieve results at their own pace. This approach can be useful when data consumers are not synchronized with data producers. The choice of the best strategy should consider factors such as the nature of the data, the processing requirements, the volume of data, and the desired level of real-time processing. In many cases, a combination of strategies may be used to manage data efficiently and meet performance requirements. It's essential to assess the trade-offs between latency, throughput, and resource utilization when deciding on the best approach for your specific use case.

< Important RE has to select data storage we are going to use in this app>

# 5 . Implementation

Application will be implemented in Java 8 coding language along with android specific java classes and complaint libraries.

# 6 . Data Matrix

Royal Enfield is going to give data matrix in DBC format for this application implementation.

# 7. Test Plan

For the Vehicle Aggregator Application Separate Test Plan will be provided with all the test case list separately. This will include Unit, Integration, System, Performance Testing

# 8.Bibliography

* SOR\_EV\_SW\_development – Veeraju, Venkat, & Tarun
* Data Aggregator.docx - Lakshmi Patel
* Data Aggregator - Explanation.pdf – Lakshmi Patel
* Android Development Guide

<https://developer.android.com/topic/architecture>