CVII Tech Stack Terminology

(proposed)

State Storage / Data Store / Embedded Database

A component storing values for VSS signals in the embedded in-vehicle space.

This component may provide values to a Data Server.

It could be a central component in a data/event architecture within one ECU (distributing data to processes / software components). In those cases it is typical to have a direct shared memory or other internal "IPC" connection, not *usually* needing a IP network.

Some databases might have some sort of "notification" feature that is used by other components directly. Because if a Data-Server supports the subscription functionality towards its external clients, it would benefit from getting notifications from the StateStorage when a new value is received there, so that internal polling is not necessary.

However, if the component implements any more logic beyond simple subscription (notify me whenever a new value is written for this data item) then it should likely be named **Data Server**, or **Event framework component** (see definitions)**.**

Some implementation options:
Apache IoTDB, REDIS, (sqlite), in-memory-table

Data Server

A component that provides VSS data on-demand or with other features (subscription to changes, or a streaming/data-flow functionality). A Data-Server primarily serves "remote" subsystems, for example a vehicle providing data server to a client in the cloud, or a data server running on one ECU providing data to another ECU.

It could however be used in a in-vehicle system, for example between ECUs. The most common connection between these subsystems that would be an IP-network or similar.

A **Data-Server** differentiates from the **State Storage**

1) A **Data Server** does not need to have its own persistent or semi-persistent data storage, which is a mandatory function of the State Storage, of course.
2) If **State Storage** has the ability for simple data interaction (e.g. get, set, and in some cases distribute data update notifications), then the Data Server has a more featureful data protocol. Examples of such protocols are VISS, GraphQL, MQTT, and various data-streaming frameworks.
3) A Data Server is often tailored towards talking to "remote" systems whereas the StateStorage more often is in a more local context (e.g. its clients execute as processes on the same operating system kernel, SoC or one ECU.

Data Client

A component that fetches data from a **Data Server** according to the chosen protocol. It typically interfaces with a database, OEM database or other, where it stores fetched data.

Event Framework Component

A component that implements logic to receive, process and redistribute events in an event-driven architecture fashion. It must implement some kind of logical operations beyond simple "subscribe to all updates of this signal", and "notify all subscribers that this data changed". If it has conditions like "Notify if *this* AND *that* event happened, and if signal value A < 3", then it is an event framework component.

In some cases the implementation of event frameworks have included some of the functionality of a data-server, data-client, etc. (see Variations)

VSS-Feeder

A typically small component that is an implementation of a data source, and a binding/translator to/from non-VSS domains:

1. It writes VSS signal data to a State Storage.
2. It consumes data from non-VSS data sources and *converts* it to VSS-defined signals before writing into State Storage.
   The VSS signals are defined by one/more VSS catalog(s) (=The shared VSS standard catalog and/or additional catalogs). The non-VSS format can be any data source and typically requires some kind of definition catalog as well. Finally, a conversion-mapping information between the non-VSS and VSS data definitions would be required.
3. A feeder may connect to *specific technologies* where non-VSS data is provided. Example: A CAN-bus VSS-feeder, ingests CAN signals, converts to appropriate VSS catalog signals, and write those into a State Storage.
4. While it is less common, the same feeder component can of course handle "written" signals as well (data is *read* from state storage, *written* to the non-VSS domain).

OEM Database (a.k.a. Cloud database, a.k.a. Data Lake)

A cloud-oriented database oriented towards storing large amounts of data typically from many sources (many vehicles) in order to provide views into the the whole set to OEM data clients.

It is operated/controlled by OEM.

Implementation: Full-scale databases, either SQL, No-SQL, or Time-Series oriented. Depending on scale, it may of course involve also massively parallel and distributed database technologies.

Neutral server Database (a.k.a. Cloud database, a.k.a. Data Lake)

Operated/controlled by a Neutral Server Company a.k.a. Data Broker.
Technology-wise we expect similar options as for the OEM database description.

3rd party application

subtypes:

3rd party cloud application

Some individually developed program that uses the vehicle data. Typically such an application is connected to OEM cloud or Neutral Server cloud.

3rd party in-vehicle application

May be connected directly to a Data Server, for example in the case of the vehicle's system allowing 3rd party applications. Those might be executing in-vehicle on an embedded Android Automotive based IVI system or similar.
It also includes applications running on a personal/mobile device that has been connected to the vehicle.

Whereas some of the components mentioned here can be mixed-and-matched with *some* technology variation,
3rd party applications are in particular need of a single (ideally) API that is consistent across all implementations.

Potential Todos. Define:
Access-control related components.
Software component repositories (for SOTA) and software deployment controllers.
Abstract protocol names
...

Note on Design Variations, consolidation/splitting of these functions.

These component names are to facilitate discussion and description of SW architecture, but do not strictly put limits on the design. In other words, a system is not required to be strictly partitioned into all of these named parts.

There are many variations to the design depending on the computing environment constraints, preferences, and the chosen implementations. Example: Some implementations of component type A might already have some of the features of component type B included, and they may as well be leveraged. The functionalities that are named above might be deliberately combined into a single software component or project, when this is warranted.

Examples:

- The **VSS feeder** might be a "plugin" to some framework, e.g. the **State Storage** database, instead of being a separate process.
- If only the latest-and-greatest value for each signal needs to be remembered, then the **State Storage** might not need to be a full "database" component but simply an in-memory table, which could then just as well be integrated into the **Data Server** code.

Despite these variations, a shared name and understanding of what each functionality is about, will simplify describing the responsibility of each component in any particular chosen setup.