Towards CVII pluggable components

Introduction

Looking at the CVII tech stack landscape (which includes input from multiple projects led by companies and organizations) I see a lot of diversity. Some diversity is good but it appears a bit disorganized.

Goal setting

- Aim to produce PoC to prove and test out uncertain areas
- Aim to produce close to production-ready implementations for areas that are not uncertain
- Agree upon requirements for real-world (production) usage
    - Preferred technologies, programming languages and runtimes
    - Non-functional requirements (performance, environments)
    - Development support requirements (CI / static-analysis tools, etc.)
    - Quality improving requirements (testing etc.)

Are current implementations too monolithic?

Can we build an ecosystem of pluggable components (as discussed many times), and in the process accelerate the creation of complete systems as well?

Current challenges

(discuss)

- Not a clear enough focus on reusable components.
- A wide diversity of programming languages and runtimes
- Not identifying what the requirements are for production-level code
- Not identifying where we have "formal" (defined/documented) interfaces as opposed to internal interfaces.

I'd like our community to start discussing some of these aspects to build a more homogeneous approach to the development.

Questions
(answer / discuss)

A) Have we agreed on the list of required components, their responsibilities and interfaces?

Tech Stack overview

B) What, in your view, are acceptable choices (and preferred choices)
 for programming languages and runtimes:
 - In-cloud
 - In-vehicle

	Today's ECUs	**Next-stage ECUs (domain controllers, central computers)**	**Cloud (computers not in-vehicle)**	**Development tools, code-generators, converters etc.**	***"Vehicle cloud computing"*** ***(think about next-next gen)***
Python					
Go					
C					
C++					
Rust					
Javascript / NodeJS					
Java					
.NET, C#, ...					
Haskell, Erlang, ...					

C) In existing architecture pictures (e.g. CCS), and framework implementations (e.g. iot-event-analytics/vehicle-edge/KUKSA.val, AOS) which interfaces:

- are documented ?
- need to be documented ?
- do NOT need to be documented (= internal / implementation detail) ?

Inspiration from these architecture pictures and other pages:

- CSS client / WAII server image in README
- CCS Reference Architecture
- Analysis of IoT-event-analytics and Vehicle-Edge projects

Completion of technology definition / implementation			
Component Name	Programming language / runtime	**Required/consumed interfaces**	**Provided Interfaces**
Kuksa.VAL:	Python	Uses various bindings, socketCAN, ... VISS-like interface also for writing data into KUKSA?	VISS protocol(?) other?
VSS Feeder (multiple)		= *StateStorage db interface*	*depends on input type (CAN, etc.)*
State Storage	SQLite database	-	SQL
OVDS Client (ccs client) (repo:ccs-w3c-client)	**Go**	VISS protocol	
OVDS Server (repo:ccs-w3c-client)	**Go**	= *StateStorage db interface*	
(CCS): LiveSim (repo:ccs-w3c-client)	**Go**	Data in OVDS Database format	
WAII:Service Manager	**Go**	(internal)	(internal)
WAII:Server Core	**Go**	(internal)	(internal)
WAII:AGT	**Go**	-	(internal) HTTP?
WAII:AT	**Go**	-	(internal) HTTP?
WAII:MQTT	**Go**	(internal, VISS-like communication)	-
WAII:WebSocket	**Go**	(internal, VISS-like communication)	-
WAII:HTTP	**Go**	(internal, VISS-like communication)	-
OVDS Database	SQLite. Probably should be production-level time series database		