Ref. Ares(2025)8661126 - 12/10/2025

Hardware Abstraction Layer for a European Software Defined Vehicle Approach

D3.1 – Modelling and simulation conducted for open-source implementations

Robert Bosch GmbH

09.10.2025

Co-funded by the European Union

Chips JU

Deliverable	D3.1 – Modelling and simulation conducted for open-source implementations
Work Package(s)	WP3
Task(s)	Task 3.1
Dissemination Level	Public
Due Date	30.09.2025
Actual Submission Date	09.10.2025
WP Leader	Dirk Slama
Task Leader	Dirk Slama
Deliverable Leader	Dirk Slama
Contact Person	Dirk Slama
E-mail	dirk.slama@bosch.com

Document History		
Revision	Date	Description
V0.1	27.08.2025	Initial version
V0.2	05.09.2025	Version reviewed, ready for PSB review
V0.3	26.09.2025	Refinement and adding more technical details
V0.4	29.09.2025	Version for PGA review
V1.0	09.10.2025	Final version for submission

Authors	
Name	Partner Short Name
Dirk Slama	Robert Bosch GmbH
Pascal Hirmer	Mercedes-Benz Group AG
Hannes Fuchs	AVL GmbH
Julio Leyva	3DS GmbH
Twan Basten	Eindhoven University of Technology
Xiangwei Cheng	Ferdinand-Steinbeis-Institut

Co-funded by the European Union

Chips JU

This document and the information contained within may not be copied, used or disclosed, entirely or partially, outside of the HAL4SDV consortium without prior permission of the project partners in written form.

Acknowledgement

The project is co-funded by the Chips Joint Undertaking (Chips JU) and National Authorities under grant agreement n° 101139789.

Disclaimer

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or National Authorities. Neither the European Union nor the granting authorities can be held responsible for them.

Table of contents

1 Summary ... 7

1.1 Introduction: HAL4SDV (https://www.hal4sdv.eu/) ... 7

2 "Mix & Match" for HAL4SDV ... 10

2.1 Introduction ... 10

2.2 Implementation ... 10

2.2.1 "Mix & Match" ("m&m") Approach ... 10

2.2.2 "m&m" in the context of HAL4SDV Landscape ... 11

2.2.3 Technical Architecture ... 12

2.3 Installation Guide ... 14

2.3.1 Required Hardware Setup ... 14

2.3.2 Installation Steps ... 14

3 Conclusion ... 21

4 References ... 22

List of figures

Figure 1: The countries involved in the HAL4SDV Consortium ... 7

Figure 2: HAL4SDV structure ... 8

Figure 3: Structure of the Transversal Activities ... 8

Figure 4: V-model-based Work Package approach ... 9

Figure 5 : What "m&m" supports ... 11

Figure 6: "m&m" in the HAL4SDV landscape ... 12

Figure 7: Technical Architecture ... 13

Figure 8: What is openDuT [4] ... 14

Figure 9: CARL Installation ... 15

Figure 10: CARL Web UI ... 16

Figure 11: Peer Devices ... 16

Figure 12: Add Network Interface ... 17

Figure 13 : Cluster Creation ... 17

Figure 14: Cluster Deployment ... 17

Figure 15: HAL4SDV Playground ... 19

Figure 16: Add SDV Runtime to Playground ... 20

Figure 17: Process of adding specific SDV Runtime ... 20

Figure 18: Select newly created SDV Runtime ... 20

Figure 19: Deliverables of WP3 21

Abbreviations

CAN	Controller Area Network
CARL	Control and Registration Logic
DBC	CAN Database File
DUT	Devices Under Test
EC	European Commission
ECU	Electronic Control Unit
EDGAR	Edge Device Global Access Router
EU	European Union
gRPC	High-performance Remote Procedure Call
HAL	Hardware Abstraction Layer
HW	Hardware
MBSE	Model-Based Systems Engineering
m&m	Mix & Match
PC	Personal Computer
RPi	Raspberry Pi
SDK	Software Development Kit
SDV	Software Defined Vehicle
SotA	State of the Art
SW	Software
TA	Transversal Activity
TRL	Technology Readiness Level
vECU	Virtual Electronic Control Unit
VHAL	Vehicle Hardware Abstraction Layer
WP	Work Package
UI	User Interface

Co-funded by the European Union
ChipsJU

1 Summary

1.1 Introduction: HAL4SDV (https://www.hal4sdv.eu/)

The HAL4SDV Project is the first in a series of SDV related projects funded by the European Commission Chips-Joint Undertaking and national funding authorities of the countries the partner organizations are resident. The project overall value amounts to € 64,5 Mio, the EC investment is € 17,8 Mio widely doubled by the national authorities.

The project is coordinated by Andreas Eckel, TTTech Computertechnik AG, Vienna Austria, leading a consortium of 50 partners and affiliated partners plus 10 associated partners. It consists of the leading European OEMs, Tier 1 suppliers, the major European semiconductor manufacturers, SMEs, academic partners, and foundations representing the European automotive industry and related academia (see Figure 1).

Figure 1: The countries involved in the HAL4SDV Consortium

The HAL4SDV project is structured in a matrix organization composed of 9 work packages (WP) following the classic V-Model Approach and a set of Transversal Activities (TA) that cover specific development domains according to Figure 2. The Transversal Activities are further structured by defining "Topics", whereof the building blocks are deducted (see Figure 3).

The work packages are grouped in "green developments" denoting open-source, non-differentiating and non-safety-relevant developments and "golden developments" representing Intellectual Property (IP) related, differentiating and safety-relevant developments (see Figure 4).

Co-funded by the European Union
ChipsJU

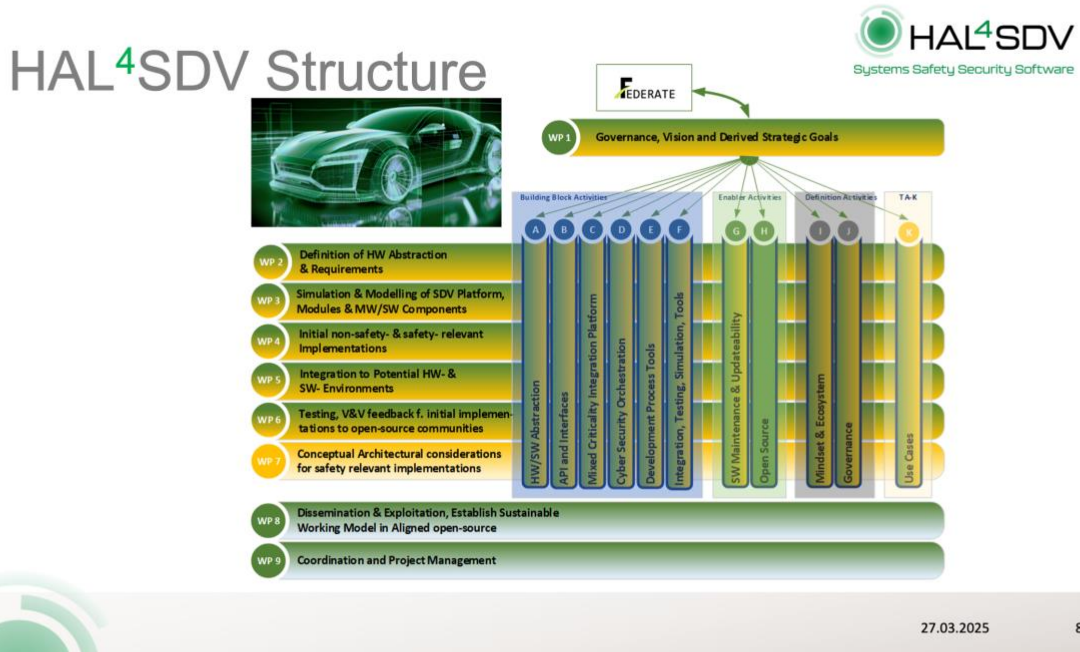


Figure 2: HAL4SDV structure

Building Block Activities

A HW/SW Abstraction	B API and Interfaces	C Mixed-Criticality Integration Platform	D Cyber Security Orchestration	E Development Process Tools	F Integration, Testing & Simulation
A01 HW Abstraction - Hypervisor	B01 DSS - Vehicle Signal Specification	C01 On-board Integration SW Environment	D01 Security Threat Analysis	E01 Linux Ecosystem for Safety	F01 Tooling for Performance
A02 Middleware besides AUTOSAR Adaptive	B02 Efficiently Integrating SDV	C02 SoA for On-board Integration SW Environment	D02 On-board Security Service Gateway SW	E02 Memory Safe Languages for Critical Systems	F02 Tool Interoperability in Automotive SW dev. Area
A03 Communication Middleware (DDS and other solutions)	B03 Mapping for Internationalization	C03 Mixed-Criticality Timing and scheduling	D03 Cloud Connectivity: Security Service Integration to On-board Integration SW Environment	E03 Open tool for architecture-Modelling following a Model-based-systems-engineering Approach for Overall Vehicle Definition	F03 Software Testing on Integration - Level
A04 Defragmentation of Interfaces	B04 Plug & Charge according to ISO Standards available as Open Implementations	C04 Shared-Memory access for On-board Integration SW Environment			F04 Virtualisation for Vehicle Subsystems
A05 Interface Concept for Service Oriented and Signal-Oriented Functions		C05 Virtualization Service for On-board Integration SW Environment			F05 Reprocessing / Replay Simulation
A06 Data Architecture for Automotive					
A07 Container/isolation for complex Applications (like HMI)					

Enabler Activities

G Software Maintenance & Updateability	H Open Source
G01 Isolation of Applications	H02 OSS Blueprints for Compliance with EU Regulations (e.g.: cyber security)
G02 Sustainable Maintenance	

Definition Activities

I Mindset & Ecosystem	J Governance
I01 Define and Show "Automotive Grade"	J01 Process Mapping: CRA Compliance with OSS
	J02 Supply Chains Open-Source Governance Model

Figure 3: Structure of the Transversal Activities

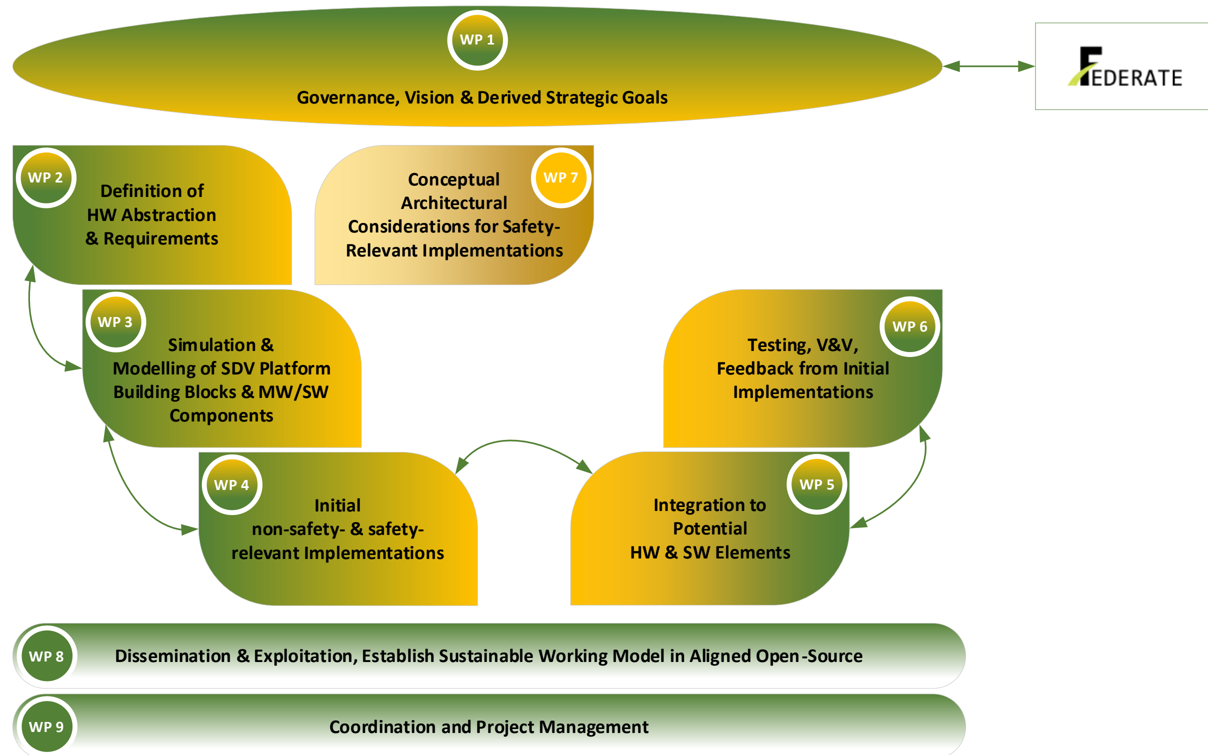


Figure 4: V-model-based Work Package approach

Co-funded by
the European Union
ChipsJU

2 "Mix & Match" for HAL4SDV

2.1 Introduction

The rapid evolution of Software-Defined Vehicles (SDVs) demands innovative testing methodologies that bridge the gap between virtual simulations and real hardware components. Traditional development workflows often struggle to integrate these domains effectively, leading to fragmented validation processes that hinder scalability and reliability. The *Mix & Match* ("*m&m*") approach emerges as a solution, offering a hybrid test environment that seamlessly combines virtual simulation components with physical hardware testing of real components. This task explores the implementation of a signal integration pipeline between OpenDUT, an open source distributed testing framework, and the HAL4SDV Playground [1], a platform for SDV hardware abstraction and simulation. The primary goal is to establish a standardized test and validation workflow, enabling engineers to evaluate SDV systems across diverse environments. By leveraging Electronic Control Unit (ECU) interfaces and cloud-based tools, this project aims to demonstrate a flexible and robust integration strategy, paving the way for future SDV development.

The *m&m* approach, as presented in the HAL4SDV project under WP3, represents a pivotal advancement in hybrid testing for SDVs. This initiative builds upon the scope defined in WP3 Task 3.1, focusing on modeling and simulation for open-source SDV platforms.

2.2 Implementation

The implementation of *m&m* leverages OpenDUT's distributed testing capabilities [2] to integrate with the HAL4SDV Playground, creating a unified platform for hybrid SDV validation. This section explores what *m&m* entails, its supported functionalities, and the underlying technical architecture.

2.2.1 "Mix & Match" ("m&m") Approach

m&m is a hybrid integration layer designed to unify access to virtual ECUs (vECUs), real ECUs, and test models, which encompass Model-Based Systems Engineering (MBSE) within the HAL4SDV ecosystem. It is delivered through integration into the HAL4SDV Playground, a collaborative environment for SDV development. At its core, *m&m* provides signal abstraction via the Vehicle Signal Specification (VSS), enabling consistent testing across diverse domains. It builds on OpenDUT's components, such as the Edge Device Global Access Router (EDGAR) and Control and Registration Logic (CARL), to manage individual device interactions and signal processing.

m&m supports a range of features (see Figure 5) tailored to hybrid SDV testing:

- Hybrid Testing Across Physical and Virtual Domains: Enables seamless integration of lightweight web prototypes, simulation models, virtual vehicle ECUs, HiL/vehicle ECUs, and SDV QEMU/ASIC embedded systems. This allows testing in mixed environments, combining virtual and physical components.
- Flexible Integration of Diverse DUT Types: Accommodates various Devices Under Test (DUTs), including vECUs and real ECUs, providing a consistent interface for validation.

Co-funded by the European Union
Chips JU

- Standardized Signal Interface via VSS: Signals are abstracted using VSS, ensuring interoperability and real-time delivery. This includes decoding CAN messages via a CAN Provider and publishing to the Kuksa Databroker for visualization in the HAL4SDV Playground.
- Integration with MBSE Systems: Offers the possibility of linking different Model-Based Systems Engineering (MBSE) systems with the HAL4SDV Playground.

These capabilities address the goal of a standardized workflow, reducing complexity in SDV development.

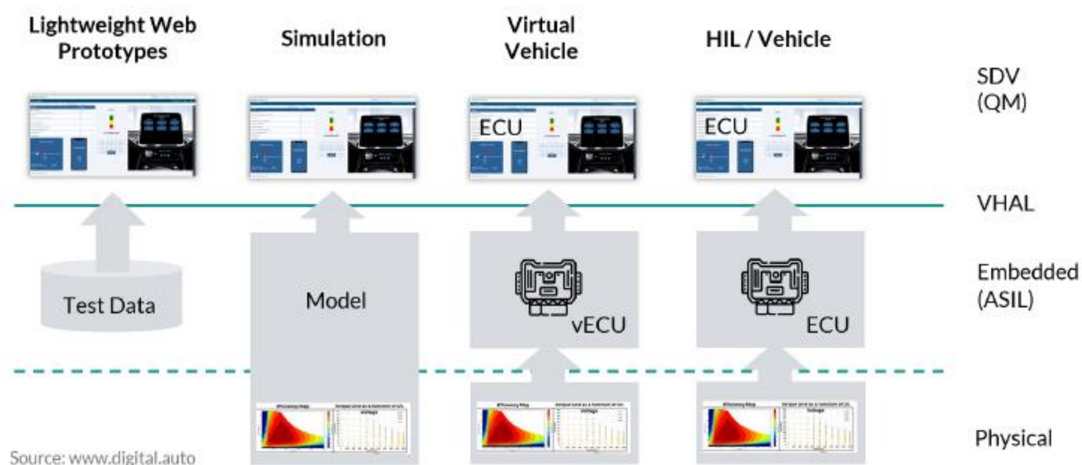


Figure 5 : What "m&m" supports

2.2.2 "m&m" in the context of HAL4SDV Landscape

In the HAL4SDV landscape, *m&m* acts as an intermediary layer (see Figure 6: "m&m" in the HAL4SDV landscape), connecting and abstracting hardware systems from the simulation on the playground.

Physical Layer: This layer represents the deployment of real ECUs and vehicles, as shown with the test vehicle setups at the Technical University of Munich (TUM) and the Technical University of Ostrava (TU Ostrava). This stage connects hardware systems directly to the *m&m* layer, ensuring that physical components can be validated using the same standardized test data and interfaces as the virtual environments.

Embedded Layer: This layer integrates safety-relevant ECU implementations within the *m&m* framework. It supports validation of embedded controllers against test data and models, maintaining consistency between simulation and hardware. This provides a pathway for ASIL-grade components to be tested alongside other DUTs.

VHAL Layer: The Vehicle Hardware Abstraction Layer (VHAL) provides standardized access to vehicle signals across models and ECUs. The VHAL layer is responsible for converting raw CAN signals into standardized VSS signals and providing hardware abstraction.

M&M Layer: The *m&m* layer acts as a mediation and orchestration framework within the HAL4SDV ecosystem. It enables distributed testing across heterogeneous components such

Co-funded by the European Union
ChipsJU

as physical ECUs and virtual ECUs, modeled using tools such as SystemC or RISC-V models and simulation models. By providing connectivity and standardized signal handling, it ensures that diverse testing setups can be integrated seamlessly into a unified validation workflow.

SDV (HAL4SDV Playground) Layer: At the top, the SDV layer is realized in the HAL4SDV Playground, offering a cloud-based environment to visualize, manage, and orchestrate tests. By interfacing through the *m&m* layer, it provides a unified access point for prototypes, simulations, virtual vehicles, and physical ECUs in one platform.

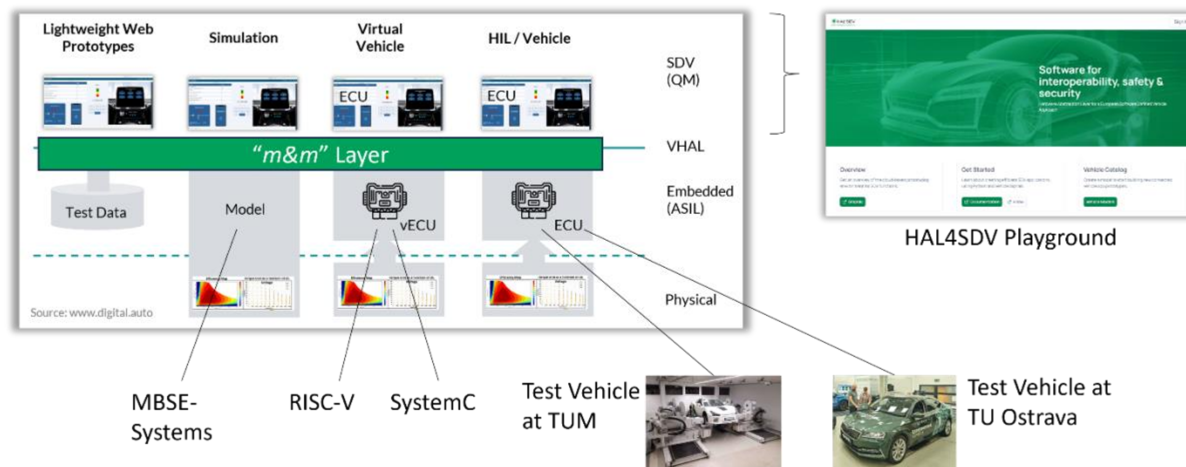


Figure 6: "m&m" in the HAL4SDV landscape

2.2.3 Technical Architecture

The technical architecture of *m&m* is built upon OpenDUT as the foundation for distributed test orchestration (see Figure 7), which is seamlessly integrated with HAL4SDV components to enable standardized signal processing, abstraction, and delivery across virtual and physical domains.

Core Components of the Architecture:

- **OpenDUT** (see Figure 8): An open framework used for automotive SW testing. This acts as an abstract layer for integrating both vECUs and physical ECU devices within the HAL4SDV ecosystem. The framework comprises CARL, which is responsible for coordinating distributed test setups, and EDGAR, which is responsible for hardware interfacing.
- **CAN Provider**: Responsible for decoding CAN messages from DUT/ECU interfaces, mapping them to the VSS format, and providing them to the Kuksa Databroker. Other providers are also available to exchange data with the Databroker or server for different data types.
- **SDV Runtime**: Includes the Kuksa Databroker as its core component, providing real-time vehicle data brokering. It handles VSS signals via gRPC/WebSocket and makes them available for consumption by applications and the HAL4SDV Playground.
- **HAL4SDV Playground**: A web-based environment that accesses VSS signals from the SDV Runtime, enabling orchestration, monitoring, and visualization of ECU testing.

Co-funded by the European Union
ChipsJU

Signal workflow:

- Real ECU devices and vECUs interface with EDGAR, which supports CAN (ISO 11898)/SocketCAN and Ethernet communication.
- Multiple EDGAR peers, e.g., Raspberry Pi (RPi), connect the DUTs to the OpenDUT framework.
- CARL serves as the backend of OpenDUT, managing DUT metadata, coordinating configurations, and establishing peer-to-peer connections via EDGAR. Test execution is configured and executed through the OpenDUT infrastructure.
- Device signals are converted into VSS format using a CAN Provider.
- VSS signals [3] are transmitted to the SDV Runtime (with Kuksa Databroker inside) via gRPC/WebSocket for real-time data exchange.
- The HAL4SDV Playground consumes these signals over gRPC/Websocket, enabling hybrid validation scenarios and simulations across virtual and physical setups.

This architecture unifies vECUs, real ECUs, and test models, providing VSS abstraction and consistent testing.

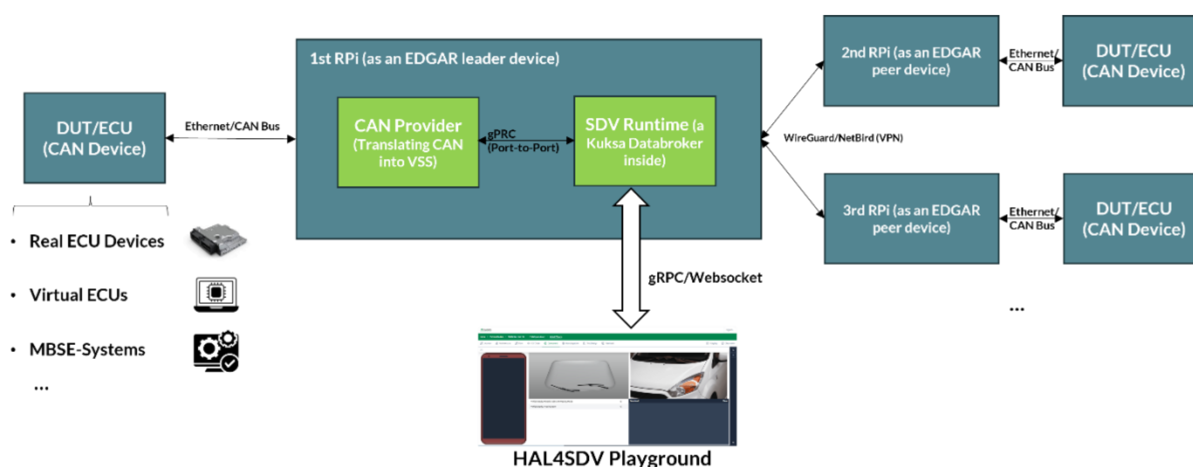


Figure 7: Technical Architecture

Co-funded by the European Union

Chips JU

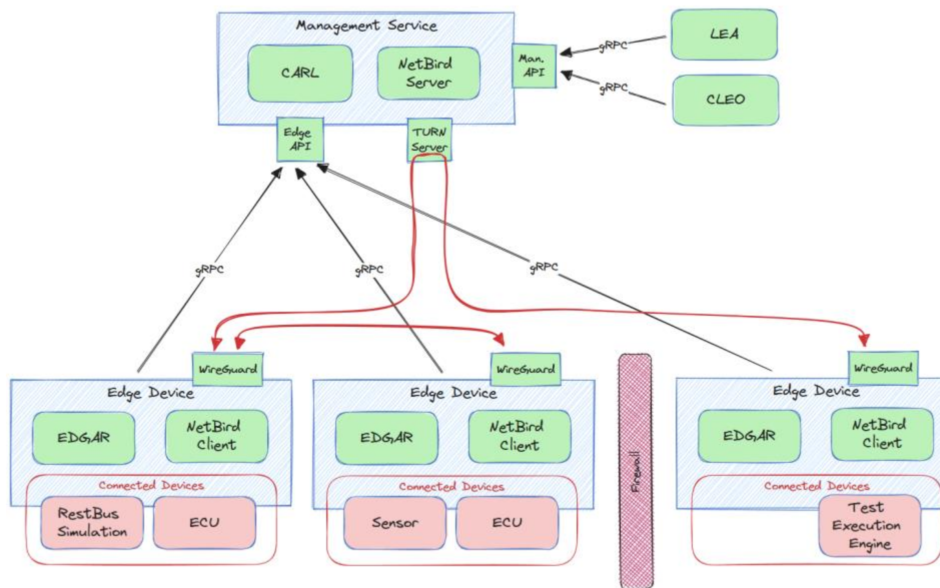


Figure 8: What is openDuT [4]

2.3 Installation Guide

This Section provides a practical installation guide for enabling *m&m* implementation in HAL4SDV. It covers package requirements, container setup, and configuration of OpenDUT and Playground components.

2.3.1 Required Hardware Setup

The following hardware is required:

- **Edge Device (e.g., Raspberry Pi 4 Model B, 4GB/8GB RAM)**

  At least one edge device acts as a leader device to run the EDGAR to connect DUTs via CAN/Ethernet

- **Device Under Test (DUT)**
  - Real ECU with Ethernet/CAN connectivity, or
  - Virtual CAN device (vcan) on Linux as a substitute

- **Host Laptop/PC**

  Runs the HAL4SDV Playground and displays the dashboard for hybrid testing

- **Linux Cloud Server**

  Hosts CARL, which is the backend service for openDuT

2.3.2 Installation Steps

Step 1: OpenDuT Setup

Co-funded by the European Union

The openDuT setup consists of setting up CARL and then setting up EDGAR for peer devices. This step is mainly divided into three parts: CARL, CARL Web UI, and EDGAR configurations.

CARL:

CARL provides the backend service for openDuT. It manages information about all the DUTs and coordinates how they are configured.

1) On the Linux cloud server, follow the instructions to set up CARL as presented in this user manual: https://opendut.eclipse.dev/book/user-manual/carl/setup.html . While editing the **/etc/hosts** file, ensure to replace the given IP address in the manual with the public IP of your cloud server.

2) After you run the **docker compose** commands, run **docker ps** to check and ensure that all the containers are up and running and are healthy. Ensure these ports are accessible. (see Figure 9)

```
aiot@aiot2:~/opendut$ docker ps
db42f98d1d92   opendut-init_keycloak                      "/provision.sh sleep"    5 hours ago   Up 5 hours             8080/tcp
, 8443/tcp, 9000/tcp                                                                            opendut-keycloak-init
e8f9d4e08413   ghcr.io/eclipse-opendut/opendut-carl:0.7.0 "/opt/entrypoint.sh …"   5 hours ago   Up 5 hours (healthy)   8080/tcp
                                                                                                opendut-carl
82cf5d1e5e3a   opendut-otel-collector                     "/otelcol-contrib --…"   5 hours ago   Up 5 hours (healthy)   4317/tcp
, 55678-55679/tcp                                                                               opendut-otel-collector
206944027dfb   netbirdio/dashboard:v2.5.0                 "/usr/bin/supervisor…"   5 hours ago   Up 5 hours (healthy)   80/tcp,
443/tcp                                                                                         opendut-netbird-dashboard
35b0e65315da   grafana/grafana:10.4.1                     "/run.sh"                5 hours ago   Up 5 hours (healthy)   3000/tcp
                                                                                                opendut-grafana
cc506c0114b5   opendut-keycloak                           "/opt/keycloak/bin/k…"   5 hours ago   Up 5 hours (healthy)   8080/tcp
, 8443/tcp, 9000/tcp                                                                            opendut-keycloak
a1206b323da4   grafana/alloy:v1.8.3                       "/bin/alloy run /etc…"   5 hours ago   Up 5 hours
                                                                                                opendut-alloy
fe833c1c4d18   opendut-nginx-webdav                       "/bin/sh -c 'nginx -…"   5 hours ago   Up 5 hours (healthy)
                                                                                                opendut-nginx-webdav
b1d361cb4ecf   traefik:v2.10.4                            "/entrypoint.sh --ap…"   5 hours ago   Up 5 hours (healthy)   0.0.0.0:
80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp, 127.0.0.1:8080->8080/tcp      opendut-traefik
c0e16261a379   postgres:14.15                             "docker-entrypoint.s…"   5 hours ago   Up 5 hours (healthy)   5432/tcp
                                                                                                opendut-keycloak-postgres
30a64a3f5c1d   grafana/loki:2.9.6                         "/usr/bin/loki -conf…"   5 hours ago   Up 5 hours (healthy)   3100/tcp
                                                                                                opendut-loki
95b1f1b3ed56   prom/prometheus:v2.51.1                    "/bin/prometheus --c…"   5 hours ago   Up 5 hours             9090/tcp
                                                                                                opendut-prometheus
2019fc9a87e6   netbirdio/signal:0.28.9                    "/go/bin/netbird-sig…"   5 hours ago   Up 5 hours
                                                                                                opendut-netbird-signal
6f9d75821da7   coturn/coturn:4.6.2                        "/coturn/entrypoint.…"   5 hours ago   Up 5 hours (healthy)
                                                                                                opendut-netbird-coturn
c2099c552629   grafana/tempo:2.4.1                        "/tempo -config.file…"   5 hours ago   Up 5 hours (healthy)
                                                                                                opendut-tempo
```

Figure 9: CARL Installation

3) Run cat the **.ci/deploy/localenv/data/secrets/. env** to find the credentials to login and access CARL.

4) Once the CARL setup is complete, the LEA web UI can be accessed by opening **carl.opendut.local** in your browser (see Figure 10). Sign in with the credentials from

the file in Step 3. Ensure that the **/etc/hosts** on your system is also modified with the public IP of the cloud server.

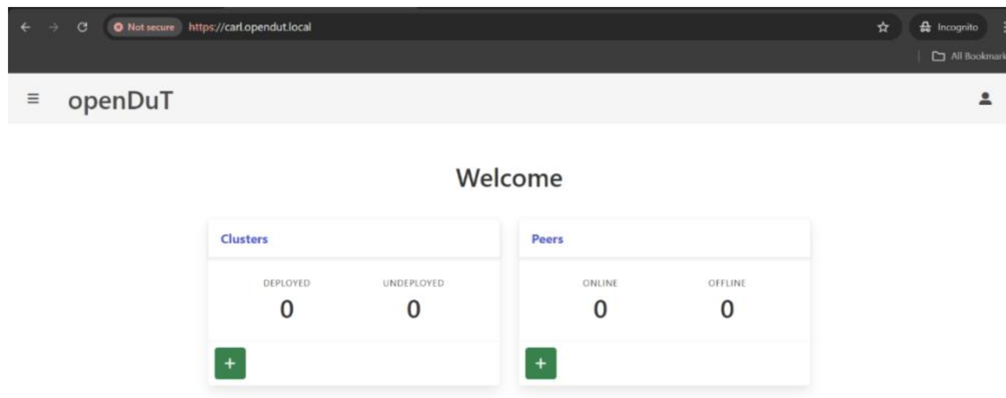


Figure 10: CARL Web UI

CARL Web UI:

1) Click on **Add Peers** and add your Peer devices that are going to be connected to ECUs. (see Figure 11)

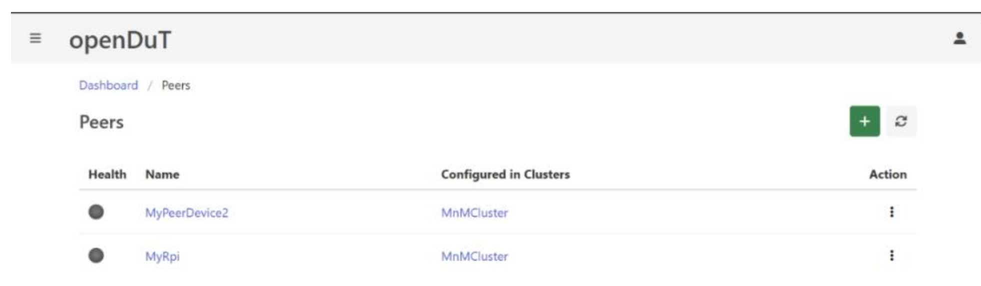


Figure 11: Peer Devices

2) For each peer device, add a network interface to set up a communication bridge. The interface can be either Ethernet or CAN (see Figure 12). For each peer device, a setup string can be generated from the Setup tab, which will be used to connect EDGAR to CARL.

Co-funded by the European Union
Chips JU

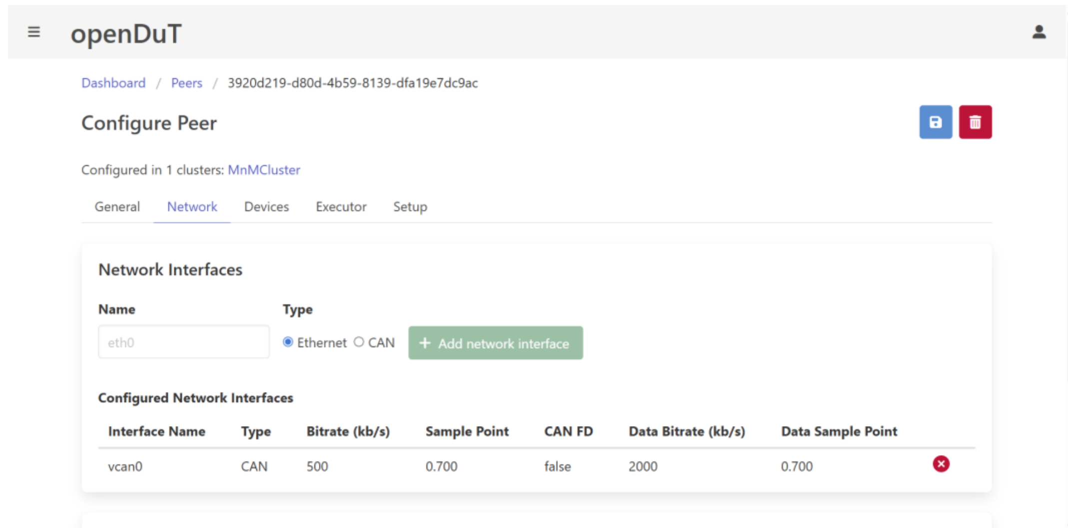


Figure 12: Add Network Interface

3) Once all Peer devices are added, click on **Create cluster**. Add the devices needed and assign a leader to each device. (see Figure 13)

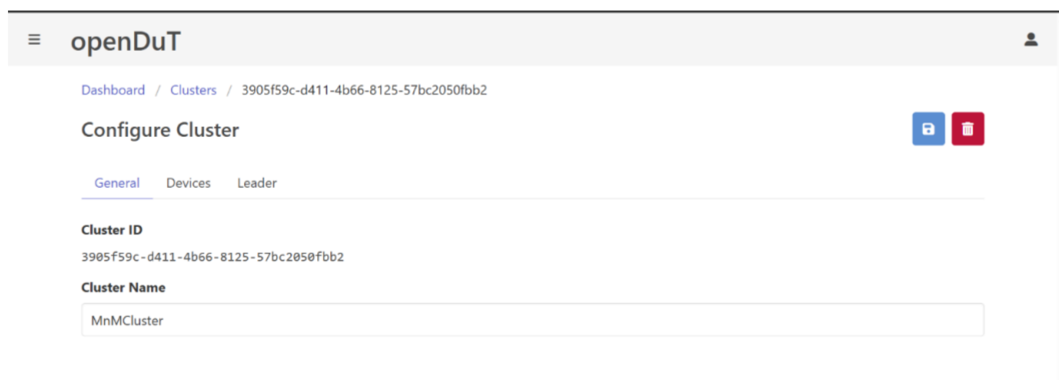


Figure 13 : Cluster Creation

4) Deploy the cluster to make it active. (see Figure 14)

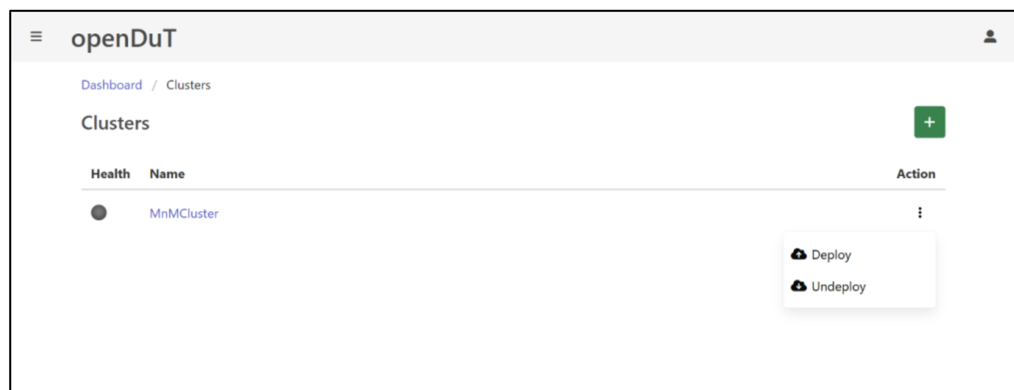


Figure 14: Cluster Deployment

EDGAR:

EDGAR hooks your DuT up to openDuT. It is a program to be installed on a Linux host, which is placed next to your ECU. A single-board computer, like a Raspberry Pi, is good enough for this purpose. Within openDuT, EDGAR is a Peer in the network.

1) On your EDGAR device, download EDGAR by following the steps mentioned in this user manual: https://opendut.eclipse.dev/book/user-manual/edgar/setup.html.

2) Once CARL and EDGAR are set up, the peer devices can communicate through a CAN or Ethernet network. You can find the detailed implementation guide here: https://opendut.eclipse.dev/book/user-manual/edgar/setup.html#can-setup.

3) After communication is established between the peer devices, these CAN signals are then sent to the CAN Provider setup via the CAN channel.

Step 2: SDV Runtime Setup

The SDV Runtime is delivered as a Docker container, offering a ready-to-use environment for SDV development that connects natively to the HAL4SDV playground. It bundles essential components such as the Kuksa Databroker, VSS support, and the Velocitas SDK, providing developers with a streamlined setup for prototyping and testing.

As an example, the following describes the installation steps on an EDGAR device running an ARM64 system.

Run the container with port forwarding for Kuksa Databroker:

- Command: `docker run -d -e RUNTIME_NAME="MyRuntimeName" -p 55555:55555 ghcr.io/eclipse-autowrx/sdv-runtime:latest`

You can replace “MyRuntimeName” with a custom name to personalize your runtime instance. This setup ensures that the Kuksa Databroker can be interacted with from outside the container (via port 55555) using the **kuksa-client** Python library.

For additional advanced options (e.g., multi-architecture builds, local development, or runtime manager integration), please refer to the official GitHub documentation: https://github.com/eclipse-autowrx/sdv-runtime

Step 3: CAN Provider Setup

The CAN Provider is delivered as a Docker container that bridges CAN bus messages with the Kuksa Databroker, translating signals defined in DBC (CAN Database File) files into the standardized VSS format. The project is available at: https://github.com/eclipse-autowrx/dreamKIT/tree/main/services/dreampack-HVAC-CAN-provider

As an example, we demonstrate the installation of the CAN Provider on an EDGAR device running an ARM64 system with virtual CAN. For other environments or more advanced configurations, please consult the official GitHub documentation.

1) Run the CAN Provider (local build & start):

- Command: ./build.sh local && ./start.sh local

2) Test the CAN Provider with kuksa-client:
   - Command: kuksa-client grpc://127.0.0.1:55555
   - Example: setTargetValue Vehicle.Body.Lights.Beam.Low.IsOn true
3) Monitor Virtual CAN traffic (vcan0):
   - Command: candump vcan0

Step 4: Connecting SDV Runtime to HAL4SDV Playground

On the HAL4SDV Playground (See **Error! Reference source not found.**), SDV Runtime can be connected to provide access to various VSS signals and their corresponding values. This enables monitoring and validation of both simulated and real ECU data in a unified environment.

The following are some concrete steps to establish the connection:

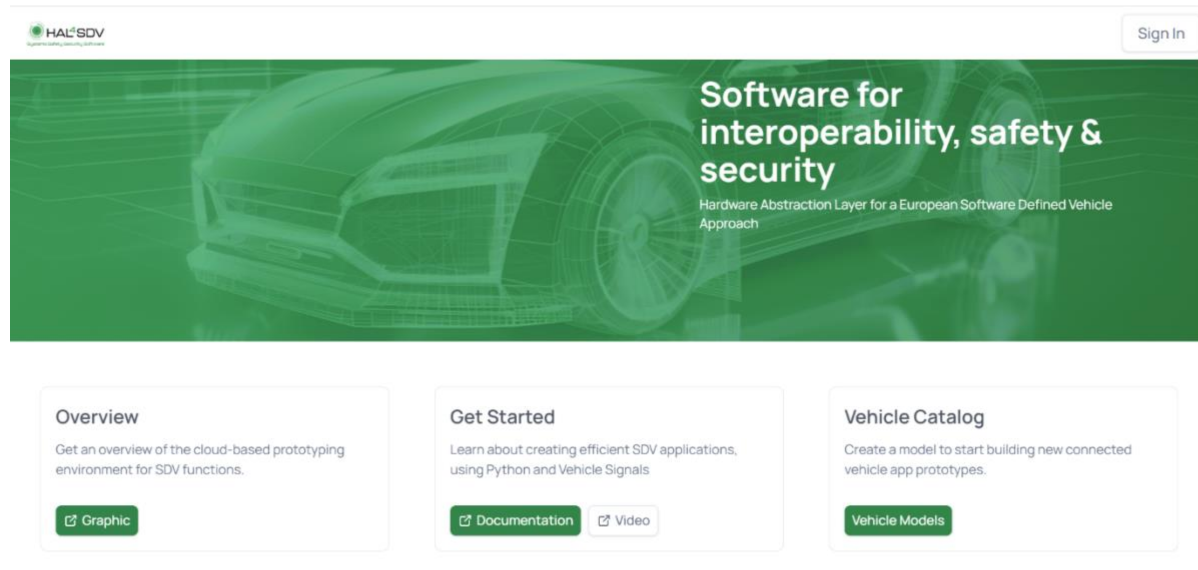


Figure 15: HAL4SDV Playground

1) Open the HAL4SDV Playground (https://hal4sdv.digital.auto/) in the browser on a Laptop/PC. (see Figure 15)
2) Navigate to the SDV Code or Dashboard tab on a Vehicle Model. Click on **Add Runtime** (see **Error! Reference source not found.**Figure 16)

Co-funded by the European Union

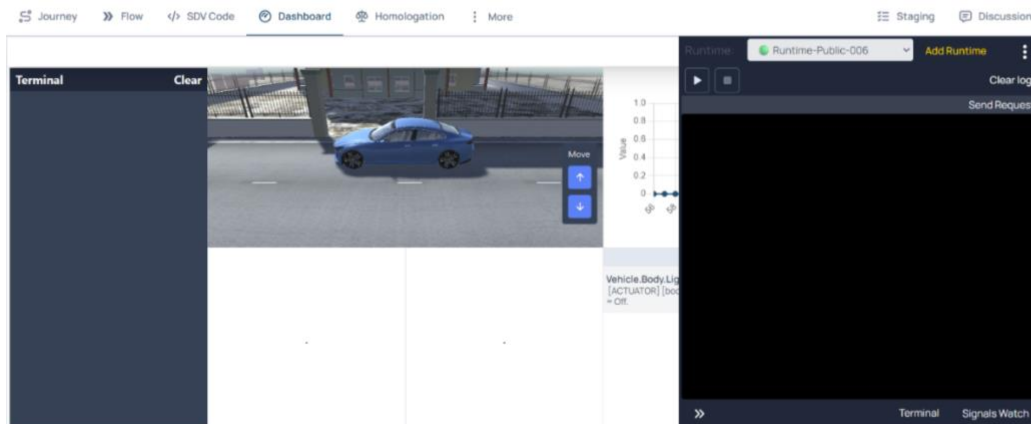


Figure 16: Add SDV Runtime to Playground

3) Add your SDV Runtime by giving the runtime name and clicking on **Add**. (see Figure 17)

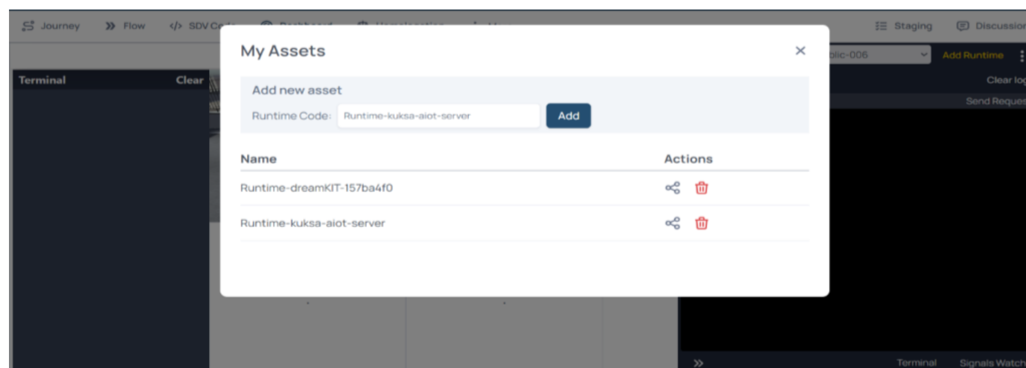


Figure 17: Process of adding specific SDV Runtime

4) Now, when you navigate back to the Runtime dropdown, you should see the newly added SDV Runtime (see Figure 18). With this, you can successfully connect to the HAL4SDV Playground. You can then visualize or simulate your ECUs or vECUs on the dashboard.

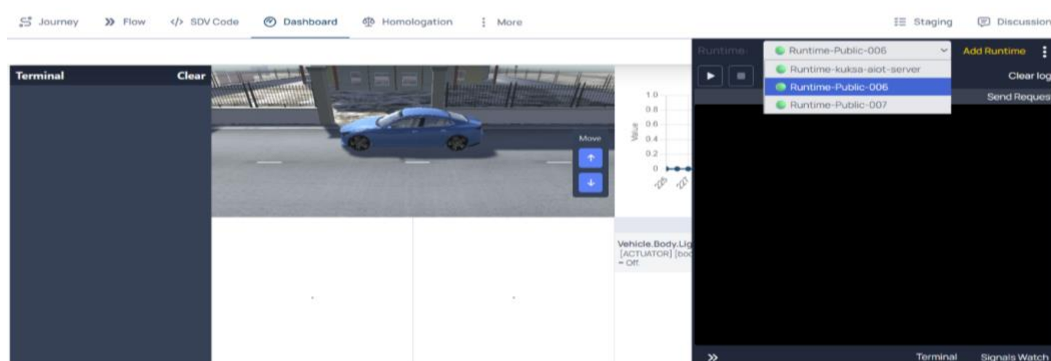
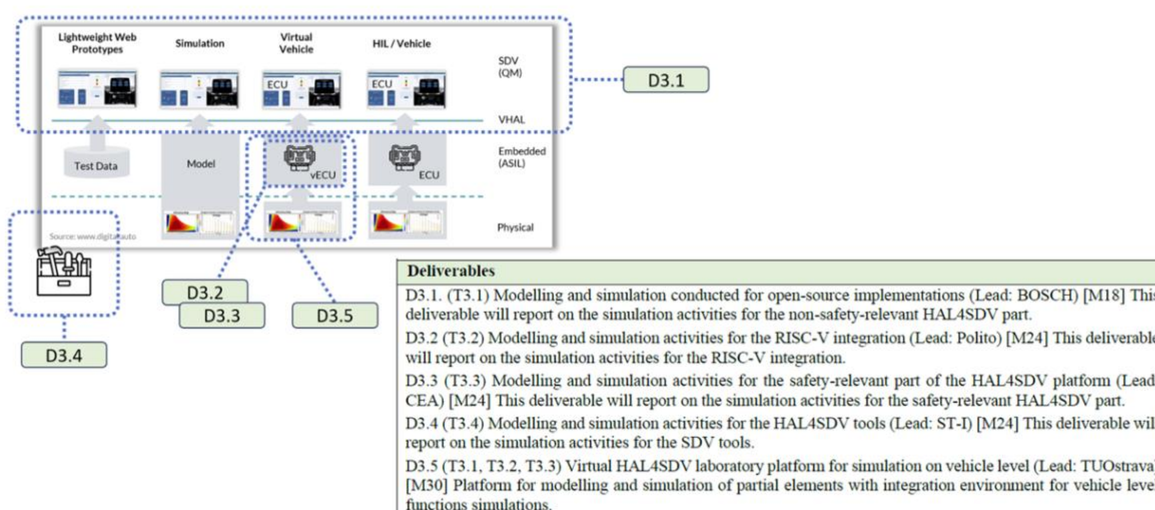


Figure 18: Select newly created SDV Runtime

Co-funded by the European Union
Chips JU

3 Conclusion

This work, Task 3.1, provides the baseline integration framework on which the different WP3 tasks can build (see Figure 19). By linking lightweight prototypes, simulation environments, virtual ECUs, and physical vehicle testing into a coherent workflow, the *m&m* approach ensures that deliverables such as RISC-V integration (D3.2), safety-relevant HAL4SDV validation (D3.3), tool-based simulation activities (D3.4), and vehicle-level laboratory platforms (D3.5) share a common foundation. In doing so, it not only supports individual tasks but also strengthens their interconnection, enabling WP3 to progress on a unified technical base.



Deliverables
D3.1. (T3.1) Modelling and simulation conducted for open-source implementations (Lead: BOSCH) [M18] This deliverable will report on the simulation activities for the non-safety-relevant HAL4SDV part.
D3.2 (T3.2) Modelling and simulation activities for the RISC-V integration (Lead: Polito) [M24] This deliverable will report on the simulation activities for the RISC-V integration.
D3.3 (T3.3) Modelling and simulation activities for the safety-relevant part of the HAL4SDV platform (Lead: CEA) [M24] This deliverable will report on the simulation activities for the safety-relevant HAL4SDV part.
D3.4 (T3.4) Modelling and simulation activities for the HAL4SDV tools (Lead: ST-I) [M24] This deliverable will report on the simulation activities for the SDV tools.
D3.5 (T3.1, T3.2, T3.3) Virtual HAL4SDV laboratory platform for simulation on vehicle level (Lead: TUOstrava) [M30] Platform for modelling and simulation of partial elements with integration environment for vehicle level functions simulations.

Figure 19: Deliverables of WP3

The *m&m* approach within HAL4SDV represents a transformative step in hybrid SDV testing, integrating OpenDUT with the HAL4SDV Playground to create a flexible, standardized workflow. By addressing the need for seamless virtual-real domain combinations and supporting diverse DUT types through VSS abstraction, *m&m* enhances SDV development efficiency.

By successfully integrating ECU device interfaces with OpenDUT components on a cloud server, this task demonstrates the feasibility of a standardized test and validation workflow that bridges virtual simulations and real hardware testing. The technical architecture, leveraging OpenDUT, CAN Provider, and SDV Runtime, demonstrates robust signal integration, with a roadmap that includes containerized deployment. Future implementations will refine real-time VSS delivery, ensuring scalable, reliable validation. Overall, this task sets a foundation for innovative SDV platforms, aligning with HAL4SDV's goals.

Co-funded by the European Union
ChipsJU

4 References

[1] Slama, D., Nonnenmacher, A., & Irawan, T. (2023). *The software-defined vehicle: A digital-first approach to creating next-generation experiences*. O'Reilly Media, Incorporated.

[2] Zyberaj, D., Hirmer, P., Aiello, M. (2025). *Using Eclipse OpenDuT for Distributed Automotive Testing*. Evaluation and Assessment in Software Engineering.

[3] Aust, S. (2022). *Vehicle API and Service Catalog for Next Generation Mobility*. The Institute of Electrical and Electronics Engineers.

[4] Eclipse Foundation (2023). *openDUT Architecture.*

https://opendut.eclipse.dev/book/architecture/index.html

Co-funded by the European Union

Chips JU