Tools and Integration

[WP4; T4.2; Deliverable: D4.2 1.0]

Non Confidential

AI training using Simulated Instruments for Machine Optimization and Verification

Version	Status	Date	Page
1.0	release	2023.11.30	1/29

Document Information

Project	ASIMOV
Grant Agreement No.	20216 ASIMOV - ITEA
Deliverable No.	D4.2
Deliverable No. in WP	WP4; T4.2
Deliverable Title	Tools and Integration
Dissemination Level	release
Document Version	1.0
Date	2023.11.30
Contact	Thomas Kotschenreuther
Organization	RA Consulting GmbH
E-Mail	t.kotschenreuther@rac.de

The ASIMOV-project was submitted in the Eureka Cluster AI Call 2021
https://eureka-clusters-ai.eu/

Task Team (Contributors to this deliverable)

Name	Partner	E-Mail
Niklas Braun	AVL	Niklas.Braun@avl.com
Liuyan Huang	AVL	liuyan.huang@avl.com
Dr. Andreas Eich	LiangDao	andreas.eich@liangdao.de
Thomas Kotschenreuther	RAC	t.kotschenreuther@rac.de
Bram van der Sanden	TNO	Bram.vandersanden@tno.nl
Lukas Schmidt	NorCom	Lukas.Schmidt@norcom.de
Hans Vanrompay	TFS	hans.vanrompay@thermofisher.com

Formal Reviewers

Version	Date	Reviewer
0.7	2023.11.29	Sebastian Moritz (NorCom); Pieter Goosen (TNO)

Change History

Version	Date	Reason for Change
0.1	2023.10.06	Initial document setup
0.2	2023.11.22	Update of document structures (version, references, tables, etc.)
0.3	2023.11.23	Added References
0.4	2023.11.23	Merge with external changes
0.5	2023-11-27	Version for formal review
0.6	2023-11-28	Smaller fixes
0.7	2023-11-29	Reference correction
1.0	2023-11-30	Fixed remarks of formal review, added formal reviewers

Abstract

This deliverable provides an overview of the tools used to apply the ASIMOV method. All phases of the ASIMOV approach are considered, from aligning the Digital Twin (DT) with its Physical Twin (PT), through the Reinforcement Learning (RL) phase, to validation of the trained AI-function by performance measurements with the two use-cases in focus: The electron microscope (STEM) and the unmanned utility vehicle (UUV).
Besides the presentation of the tools, also potential extensions are described, that would help to use the ASIMOV method more efficiently.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This deliverable provides an overview of the tools used to apply the ASIMOV method. All phases of the ASIMOV approach are considered, from aligning the Digital Twin (DT) with its Physical Twin (PT), through the Reinforcement Learning (RL) phase, to validation by performance measurements.
Besides the presentation of the tools, also potential extensions are described, that would help to use the ASIMOV method more efficiently.

1.1 T4.2 within ASIMOV

Within the ASIMOV process several use scenarios are available that require data exchange with the physical system. In [1] the trained AI should be connected either to the digital twin, or the physical system to see, if the trained strategies work with the physical system.

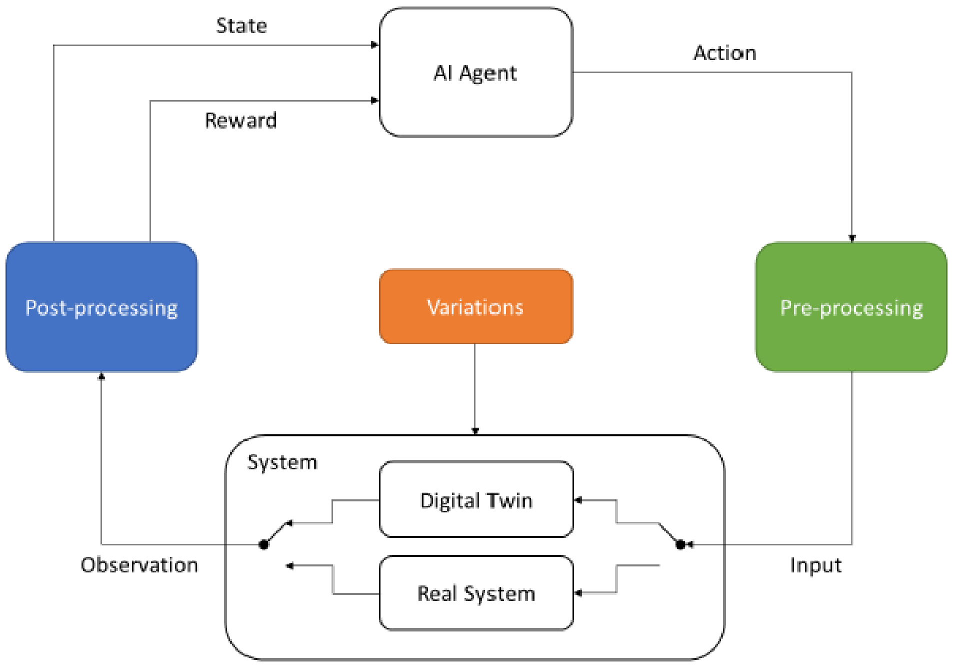


Figure 1: Connectivity of the AI agent [1]

But even in a more simulated environment (s. Figure 2) as described within [2], the physical parameters need to be read out of some physical system. And, when the functionality is finally available, a KPI based validation needs access to the physical system as well.

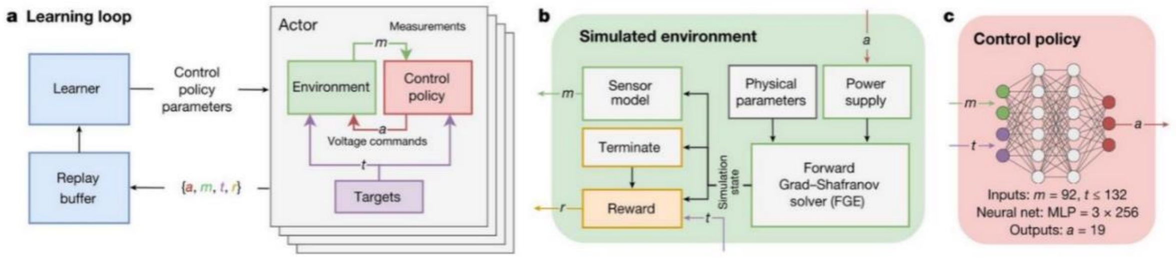


Figure 2: ASIMOV approach within a simulated environment, taken from [2]

Regardless of the intended use, data exchange between the digital and physical twin requires the use of some dedicated tools, unless the physical system is already accessible via a well-defined API, as in the case of the electron microscope. So, the tools described within this document focus on the UUV-use case.

1.2 Measurable data

One general aspect of the relation of a Digital Twin and a physical system is the understanding that both instances should operate on the same inputs, including distortions, and provide comparable results. The task of data acquisition within the physical system can be used as a measurement that could be also provided to the digital twin. With that the digital system could operate on the same data, including all the distortions that affect the physical system.
Within a first step, the measurements are done as offline measurements to have a dataset that could be analysed for basic decisions like e.g., the AI-model layout characteristics.
In a later phase, during the model validation, online measurements could be used to validate the basic behaviour of the AI-model.

Version	Status	Date	Page
1.0	release	2023.11.30	6/29

For the training phase several datasets for the training are necessary. These could be generated or also taken as offline measurements.
When the AI is trained, a final validation is necessary based on some KPI-values to validate the proper operation of the AI-model. An optional fine-tuning phase would allow to tune some parameters directly within the physical system to reach the achieved quality goals measured by KPI values (see also [3]).

2 Data exchange between the Physical and the Digitial Twin

The data exchange between either the Cyber Physical System (CPS) or Physical Twin (PT) and the Digital Twin (DT) is used within different phases of the overall process.

2.1 Digital Twin, Digital Shadow, Digital Model

Already the type of the digital counterpart of the physical system maps to specific requirements regarding the connection and the data exchange between the two instances.
Already, in D4.1 [3] the definition of Aheleroff [4] is used, where several variants of digital representations of physical systems are mentioned. They basically differ in the way of interacting with each other:

- Digital models: They behave like the physical system, but they are not coupled directly to any.
- Digital shadow: The digital shadow is a digital model, that is linked to the physical counterpart in that it receives the same input signals i.e., it is continuously updated by the physical values.
- Digital twin: The digital twin is a digital model with bidirectional coupling of signal values with the physical system. Like the digital shadow, it receives continuously the same real input signals as the physical system but feeds back the output signals.

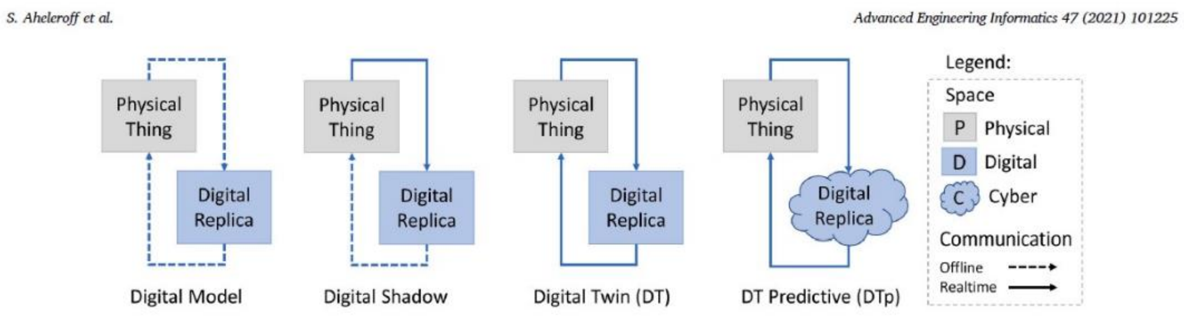


Figure 3: Relation between digital and physical part with Digital Model, Digital Shadow and Digital Twin [4] [3]

The mode of connection differs between the variants in Figure 3, as only the latter two are connected continuously. Thus, the digital model is used mainly with offline data.

In ASIMOV, the digital twin (DT) as sketched above is in the focus. Thus, all the requirements regarding the synchronization apply to the respective use cases.

The data link between CPS and DT could be separated into two independent domains:
1. The data model: Which data is retrieved from the CPS to be provided to the DT, and vice versa, which data for the CPS needs to be provided by the DT.
2. The aspects of the communication itself, respectively the requirements coming from the use case of the DT.

Within the ASIMOV approach the DT is used to train an AI functionality based on Reinforcement Learning (RL). The final goal is to apply the trained model to the CPS in the end, and thus provide the AI-functionality within the CPS itself.

2.1.1 Synchronization

One aspect of the coupling of the digital with the physical system is the synchronization. In case of a Digital Model, there's no requirement towards the synchronization as there is no continuous connection. In this case it would be sufficient to e.g., record some data from the physical system and evaluate this recording at the time of modelling.
Similarly, the Digital Model (e.g., a simulation) could provide some output data, that can be used as a prediction, or a functional reference, which could be persisted as a recording and used independently to e.g., validate some functionality within the physical system. With a plain Digital Model, the digital and physical part are not synchronized. But to have a valuable dataset, it's always necessary that the datasets are consistent i.e., they have accurate time-information, and capture the information required to reproduce the data.

Synchronized systems act in a way that some state-change on the source-side also leads to a state-change on the target side at the same time. This is only possible with discrete-time systems, where values are sampled and evaluated for a discrete point in time and all systems concerned are acting synchronously. The same is required for the evaluation of the output signals; they need to have effect at a synchronized time in the future.

One automotive protocol that supports these requirements is the ASAM XCP [5]. This protocol allows time-stamped samples of measurements to be transmitted with the sampling-event within the source system. Similarly, it provides the capability to transmit so called stimulation signals which are samples planned to be injected into the target system at a given point of time. Having both of these features supported allows to do the process of rapid prototyping, which is finally a synchronized co-simulation i.e., some functionality based on some synchronously sampled inputs is computed and the outputs are also provided in sync with the update schedule of the target system. The externally computed values override the internal ones before the values are used within the next cycle either to be sent out on some bus or used within the next computation internally.

The XCP protocol supports multiple physical layers for communication as e.g., CAN, Ethernet/IP (UDP + TCP), Serial line or FlexRay.

2.2 Phases of data usage within ASIMOV

Within the process given in [3] in ASIMOV four main phases are in focus:

1. Preparation phase:
   This phase is used to first model the digital model to be finally used for the digital twin. During this phase measurements of the physical system allow to decide about the model structure or to tune parameters of the digital model.
   The measurements used for this phase do not necessarily need to be in real-time. Mostly this will be offline data evaluated for some characteristics.

2. Training phase:
   During this phase the model of the DT is prepared, and the training data is provided. In some cases, it is possible to retrieve the training data from direct context of the CPS. Thus, the sensors of CPS reflect the context information that could be read from the CPS.
   The training data is also not real-time.
   Besides the training data itself, it must be decided, if the training reached the required quality level. For this validation it might also be necessary to measure data from the CPS directly. This could also be already a pre-validation by providing the trained parameters into the AI-model within the CPS and measuring the KPI values.

3. Validation phase
   Finally, the trained model is to be calibrated into the target system to validate its proper functionality. After deploying the model, the predefined KPI-values are measured to evaluate if the trained model behaves as expected within the context of the CPS.

4. Finetuning phase
   Sometimes it is necessary to compensate for the differences of the simulated to the real environment by finetuning some parameters in the physical system directly. This can be parameters that affect the acquisition of input data, but also modify slightly the output values.

Independent of the usage and thus the linking of the DT to the CPS, it is common to all scenarios, that the DT needs to be designed or derived from knowledge of the CPS.

2.3 Structural restrictions and requirements

2.3.1 Measured Data types

The synchronization of the DT and the CPS is based on different kind of data.

With focus on the UUV use case, measurements are related to automotive systems. The protocols used there allow to measure several types of signals, like Boolean, Integer or Float values. One characteristic for a measurement is that the measurement data needs to be related to some system state. The most common way to synchronise measurements is based on the sampling time of the signals.
Within this context, a measurement is a time series of time-stamped measurement samples.
A sample is a collection of signals related to physical system. They represent values measured or calculated within the target system e.g., speed, temperature, angles, torque, state information, etc.

In general, the measurement values have two representations within the target system:

1. The raw data format: Sensors typically provide their measurement values in encoded form e.g., as an integer value representing some physical value. Also, within ECU-SW often the signals values are encoded into a form depending on the final usage of the values. Stored values might be reduced in space while values which are used for a controlling process might be encoded in a way to be transmitted via network, or in a comfortable format for the controlling algorithm. Common is the understanding that the usage of floating-point values has some drawbacks regarding space and computation speed. Knowing that, most raw value representations are directly encoded into some integer value, or as a variant are represented as a fixed-point value.
2. Physical representation: As soon as the signals values need to be evaluated within the context of other physical values, it's necessary to convert the raw value into a representation that allows to interpret the value as a physical signal value.

In terms of performance, it is often beneficial and thus it is supported by most measurement encodings to use the raw value representation. This requires that the measurement system does not only get information how to access the related signal values, but also information about the encoding of the respective signals. The measurement system will acquire the raw values directly and do the conversion into physical signal values after transmission of the measurements into the measuring system.

More complex measurement data like e.g., object lists, LIDAR data, trajectories, point cloud data or video data could be measured as a stream. This is a practical way, as the source system needs to serialize the measurement data already to transmit it within the vehicle. The measurement system then just uses the same data stream to route the information out of the vehicle into the measuring system. As with the standard measurement values, it is also necessary to know the details of the encoding to convert the received information into some meaningful data like e.g., a video stream or a sequence of bounding boxes.

Besides the measurement of signal values there are use cases that require to modify some settings e.g., by enabling or disabling some functionality. The process that allows that is named calibration.
With this functionality it is possible to modify normally constant values and by that change or tune the target systems behaviour.

The data types used for calibration are like those used for measuring signals. All platform specific data types of the target system can be tuned. Additionally for some controlling processes it is common to use data tables to encode the controlling strategy. Thus, also these tables or maps are used as a calibration data type. They differ depending on the targeted controlling strategy from a vector, which is referred as a 1D map up to 5D-maps for the more complex strategies.

Especially within ASIMOV there's the need to transfer the trained AI data into the target system for validation. To do that, the same mechanisms could be used as for the calibration of complex data types. This has been evaluated within the respective embedded software components (3.7.3.2 or 3.7.3.7).

2.3.2 Interfaces to access data

To receive vehicle data, it is necessary to get access to the respective networks within a vehicle. In today's vehicles several network types are integrated depending on the needs of the respective functionality.
A very mature system is the Controller Area Network (CAN) bus. Via this bus the ECUs exchange data frames that contain up to eight data bytes in classic CAN systems. Each of the frames is typed by its CAN-ID and depending on that CAN-ID the frame can be parsed for the contained information. One standard method is to encode several signals per frame in some raw-data format. A description of this encoding allows to decode this information and thus make the information usable as signal values.
Newer variants of the CAN bus allow to transmit more data bytes per frame and thus increase the data rate. With CAN-FD (Flexible Datarate) the data part of a CAN frame is used with an increased frequency to pack more data bytes into the same slot. A factor of up to 8 allows to transmit a maximum of 64 Bytes instead of only 8 Bytes with the classic CAN. CAN-XL extends this even further to a maximum of 2048 Bytes within one frame. Table 1 shows this with the respective achievable data rate.

Table 1: CAN data rates

CAN Version	**Max data rate**	**Max. Bytes per frame**
CAN (classic)	2 Mbps	8
CAN-FD	5-8 Mbps	64
CAN-XL	20 Mbps	2048

CAN-bus access is realized by dedicated interfaces, a piece of hardware that allows to access the CAN-medium to receive and send CAN-messages.

Ethernet is a very well-known transport medium used with computer networks. With some modifications, this technology is also used inside vehicles as e.g., Automotive Ethernet [6]. Similarly, to standard computers Ethernet is also used within vehicles utilizing the internet protocol (IP) and thus allow TCP connections or UDP datagrams. Besides this, sometimes also raw Ethernet frames are used depending on the target requirements.
Within IP, packets might range up to 1522 Bytes per package. With the bigger data size per packet there are way more possibilities to encode measurement data. Sometimes with UDP datagrams there are similar structures used as with the CAN to encode signal values directly in dedicated locations, but due to the many protocols available for IP networking much more formats and protocols exist than for the CAN.
To access Ethernet networks standard network interfaces can be used. In some systems there might be some modifications that require to use a dedicated interface hardware.

Based on IP networks there are several protocols that are used within automotive environments. On the one hand e.g., the XCP protocol that also supports the CAN bus can also be used within IP networks. Another example is the SOME-IP protocol which is used within AUTOSAR systems to interconnect the different nodes and services.
As a middleware also ROS [7] is based on IP communication. With ROS-2 the standardized DDS [8] is supported, which is also supported within AUTOSAR networks.

Within research projects, ROS is often used as it directly supports the newer sensor types like RADAR or LIDAR. Also, many of the related data types e.g., to handle trajectories, used within the context of autonomous driving are directly supported by ROS. This makes the system very interesting for measurement access within ASIMOV (s. also 3.7.3.3).

2.3.3 Data exchange possibilities

Depending on the accessibility of the physical system, several data exchange strategies come into consideration. First of all, it needs to be separated between stationary systems and mobile systems.
Stationary systems most likely provide a wired connection which can be used to read measurement data. Mobile systems like vehicles can provide this when the measuring system can be attached to the mobile device. In this case, offline measurements (recordings) are possible e.g., by using embedded systems.
But even when it's possible to attach a measuring device to the mobile system, any kind of online measurement requires a wireless transfer of the measuring data to the data consuming instance. And in general, the wireless bandwidth and response times are worse than with wired connections.

For any kind of online measurement, it is necessary to know about the requirements regarding delays or failures. Basically, two operating modes are relevant, described as “dashboard-mode” and “forensic mode”.

Within the “dashboard-mode”, it is more important to receive the most recent values. When data gets lost due to some circumstances, it's not relevant to re-transmit the lost information, it's more important to update the data as soon as new data is available. A generic use-case for this could be a controlling algorithm, or a dashboard visualizing current signal values.

The “forensic-mode” requires to receive all data without any distortions and in the correct order. When a data loss happens, the lost data needs to be re-transmitted which could cause more load on the transmission and the related systems, due to the management of local buffers. The forensic mode would be required e.g., when a recording should be done utilizing a wireless remote connection.

3 Tools used within the Demonstrator Use-Cases

Within the ASIMOV use-cases dedicated tools for the exchange are mainly used for the vehicle related use cases. The Electron Microscope provides a Python interface and thus can be integrated in several environments quite easily without the need to use specialized tools to interact with the physical part. Due to this, the following Tools have been used with the UUV use-case.

3.1 Model.CONNECT

Model.CONNECT [9] is a co-simulation environment that enables the coupling of all the different components needed to simulate a vehicle together with its driving functions, sensors, vehicle dynamics and 3D scene environments. Thanks to its extensive scripting support, it enables the integration of Carla into a fully automated testing pipeline for conducting the 3D scenario environment simulations. In conjunction with Testbed.CONNECT [10], Model.CONNECT effectively manages timing, ensuring a consistent data flow throughout every aspect of the simulation, the generated results would later be analyzed in postprocessing and can then be later used for the Reinforcement Learning.
Model.CONNECT does also support the creation of XCP slaves as part of the supported FMU standard. This would allow to couple them to a XCP master, which could then collect measurement data not only from XCP-supporting physical ECUs, but also from their virtual counterparts.

3.2 Testbed.CONNECT

By utilizing Testbed.CONNECT to link a physical vehicle on the testbed (which is not implemented in this deliverable) with Model.CONNECT, it enables the acquisition of real performance data from the physical vehicle within the simulation (see Figure 4:). Comparing this data with the performance of the digital vehicle can contribute to further refining the digital vehicle model. It is a valuable concept that may be considered in future iterations.

As Testbed.CONNECT offers real-time computation in combination with IO support it could be used in multiple ways in the ASIMOV project. Direct real-time communication to the vehicle and a connection to the simulation allows for comparison of simulated and real data. The concept of an adaptation model could even be realized inside Testbed.CONNECT directly in form of an FMU.

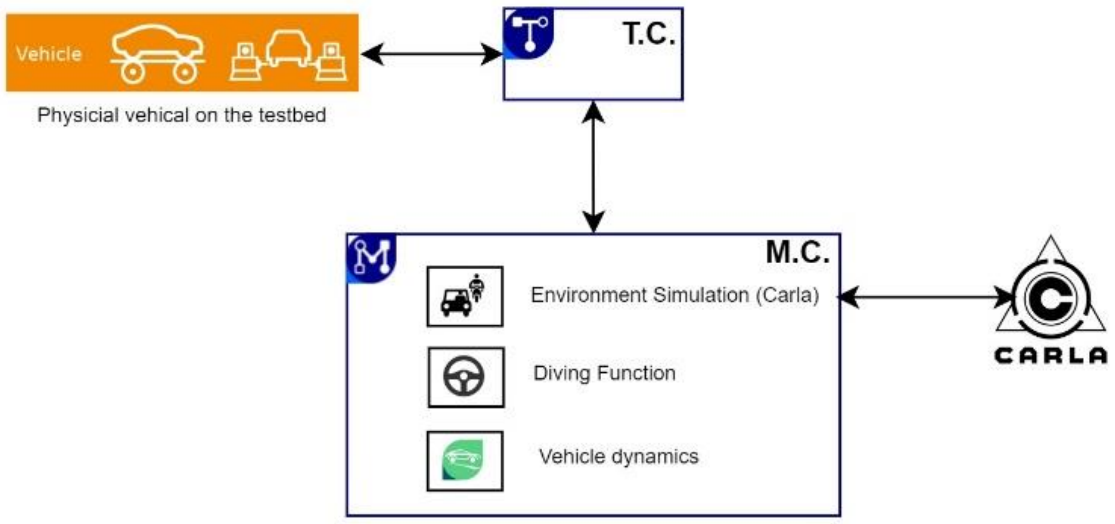

Vehicle
Physicial vehical on the testbed
T.C.
M.C.
Environment Simulation (Carla)
Diving Function
Vehicle dynamics
CARLA

Figure 4: Interaction of vehicle, Testbed.CONNECT (T.C.), Model.-CONNECT (M.C.) and the simulation in CARLA.

3.3 DiagRA-X

DiagRA-X [11] is an automotive Measurement and Calibration (MC) Tool. It provides the capability to measure signals from automotive systems as communication lines or Electronic Control Units (ECUs). The calibration refers to the capability to tune parameters within the ECUs, while monitoring the effects as a measurement. With these capabilities it is possible to optimize parameters for the particular usage of the target device.

With these capabilities, DiagRA-X can be used for several usecases within the ASIMOV approach.

- Measurements from the physical system as a basis for the model generation
  - Online measurements
  - Offline measurements (recordings)
- KPI-based measurements to validate the quality of the model.
- Tuning parameters within the physical system to force a specific behavior / enable a special functionality.
- Writing of trained parameter-sets into the target system to update (AI-)components.
- Support KPI-based measurements to support metrics-based evaluation of the parameter set.
  - Online measurements
  - Offline measurements

3.3.1 Workspace Setup

To enable DiagRA-X to interact with the target systems, a workspace setup is used. Within this workspace setup, it is defined, which devices are known to DiagRA-X and which interfaces DiagRA-X can use to interact with them.

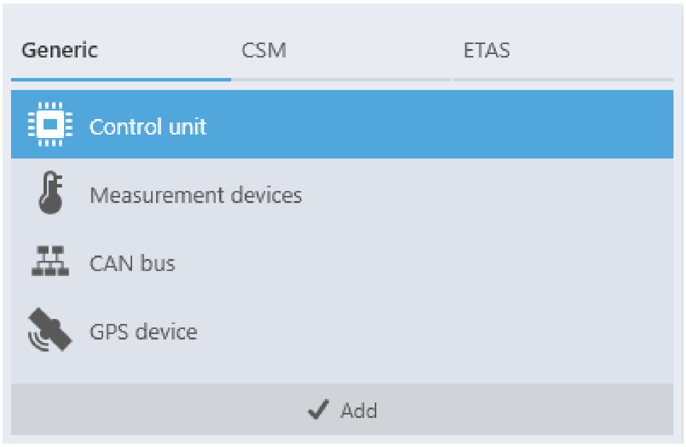
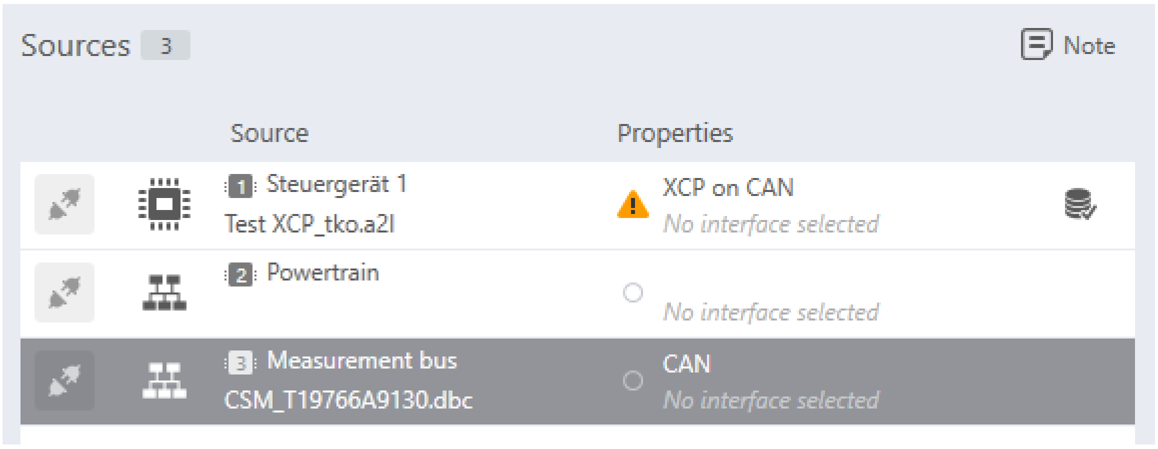


Figure 5: Selection of device type within DiagRA-X

The following types of devices are supported (s. Figure 5):

- ECUs
- Measurement Devices (CAN-based or Ethernet based)
- Busses
- GPS-devices
- Dedicated devices

The workspace devices are selected by type and have a type-specific set of parameters to set.


Sources 3 Note

	Source	Properties
1	Steuergerät 1 Test XCP_tko.a2l	XCP on CAN No interface selected
2	Powertrain	No interface selected
3	Measurement bus CSM_T19766A9130.dbc	CAN No interface selected

Figure 6: Workspace setup within DiagRA-X for an ECU and some measurement device

In case of an ECU, it would be an A2L-File, while for a CAN-bus a DBC-file is expected.

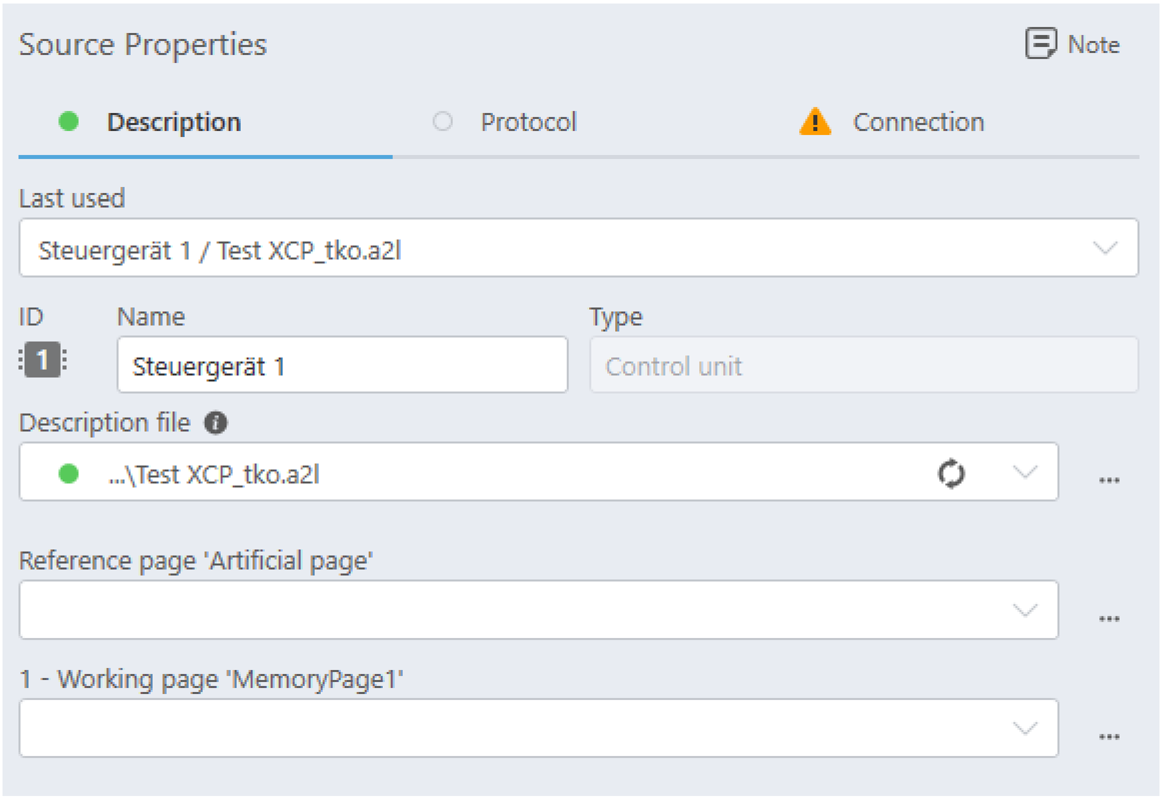
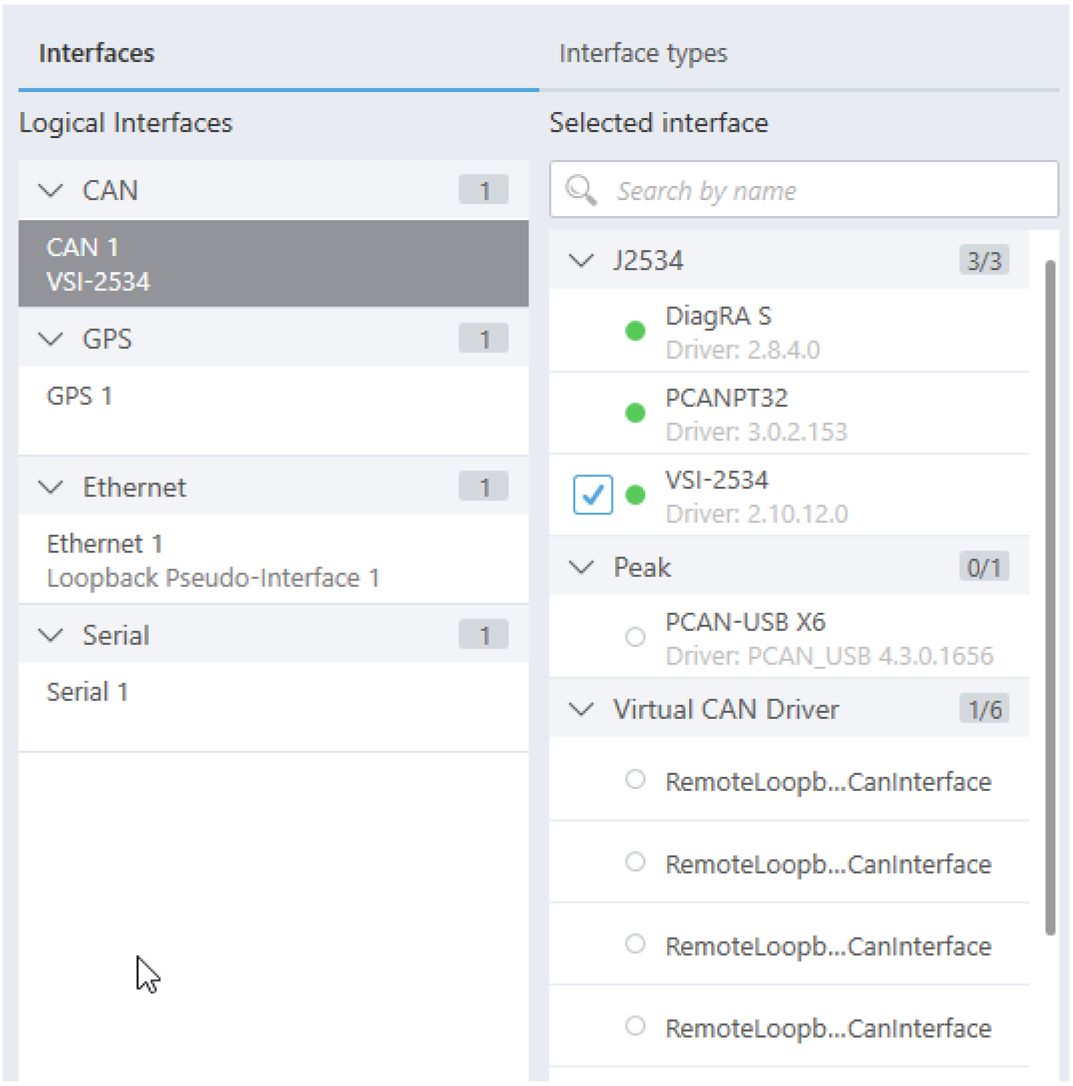

Source Properties
Note
Description
Protocol
Connection
Last used
Steuergerät 1 / Test XCP_tko.a2l
ID
1
Name
Steuergerät 1
Type
Control unit
Description file
...\Test XCP_tko.a2l
Reference page 'Artificial page'
1 - Working page 'MemoryPage1'

Figure 7: Details of parameters to specify an ECU by a standardized description file.

When the workspace setup specification is complete, the referenced interfaces can be mapped to the real interfaces available within the local system. This is used to handle the interfaces of the workspace setup as logical interfaces that are mapped to the real interfaces available at the system where the measurements are started. So, the same setup can be used at different locations. Figure 8 shows the view to define that mapping.



Logical Interfaces

- CAN 1
  - CAN 1 VSI-2534
- GPS 1
  - GPS 1
- Ethernet 1
  - Ethernet 1 Loopback Pseudo-Interface 1
- Serial 1
  - Serial 1

Selected interface

- J2534 3/3
  - ● DiagRA S, Driver: 2.8.4.0
  - ● PCANPT32, Driver: 3.0.2.153
  - ✓ ● VSI-2534, Driver: 2.10.12.0
- Peak 0/1
  - ○ PCAN-USB X6, Driver: PCAN_USB 4.3.0.1656
- Virtual CAN Driver 1/6
  - ○ RemoteLoopb...CanInterface
  - ○ RemoteLoopb...CanInterface
  - ○ RemoteLoopb...CanInterface
  - ○ RemoteLoopb...CanInterface

Figure 8: Mapping of logical to available interfaces by type.

When the hardware setup is done, the experiment setup follows.

3.3.2 Experiment definition

The experiment definition specifies, which data is measured and how it is visualized and/or recorded. Therefore, the experiment setup allows to define several Worksheets per Experiment, and on each worksheet multiple visualizers could be placed.
Besides the pure layout of the worksheets, also the assignment of the related labels is kept within the experiment definition. Thus, a worksheet can be seen as the working place to fulfil a particular task e.g., by grouping the relevant KPIs.

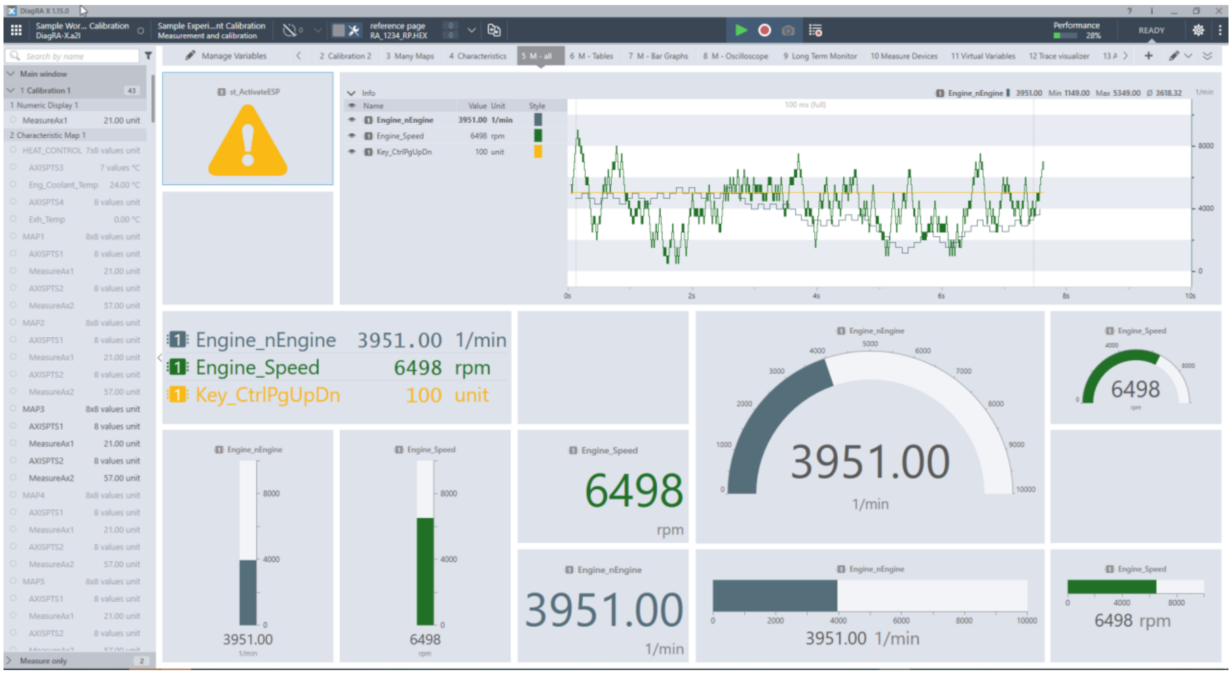


Figure 9: Different visualizer types in DiagRA-X

To have an overview of all worksheets within the current experiment, an overview is provided that allows to see a preview of each worksheet. By a double-click on a preview image, the respective worksheet will be opened.

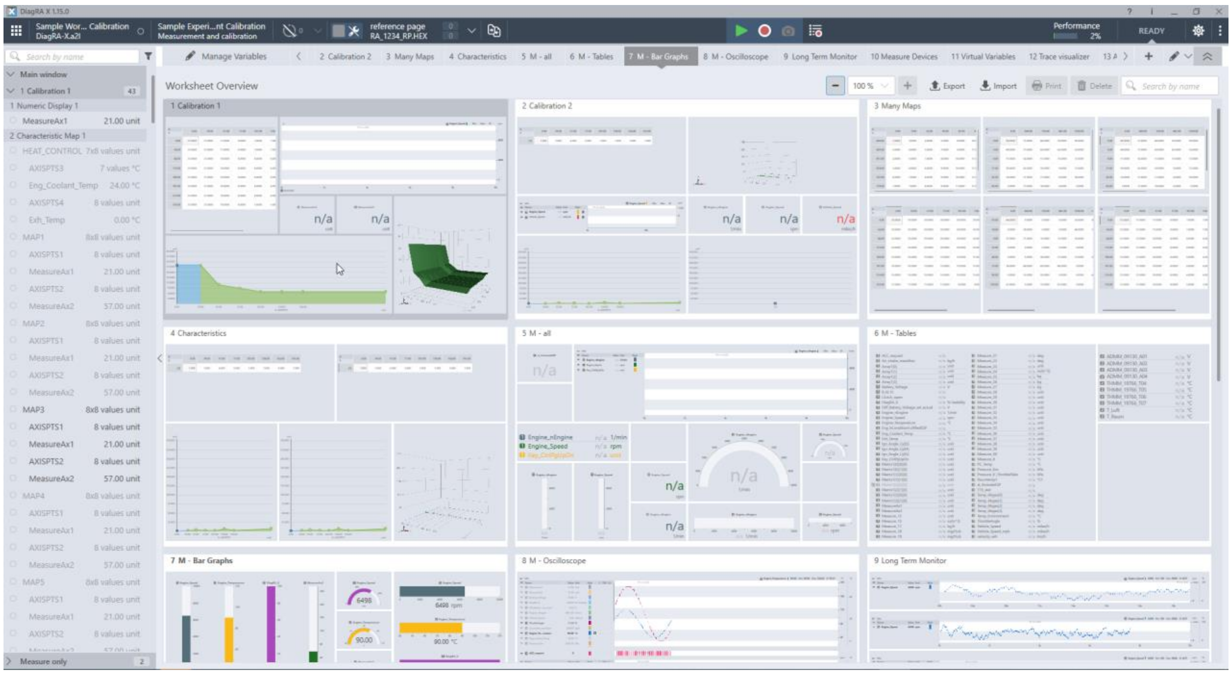
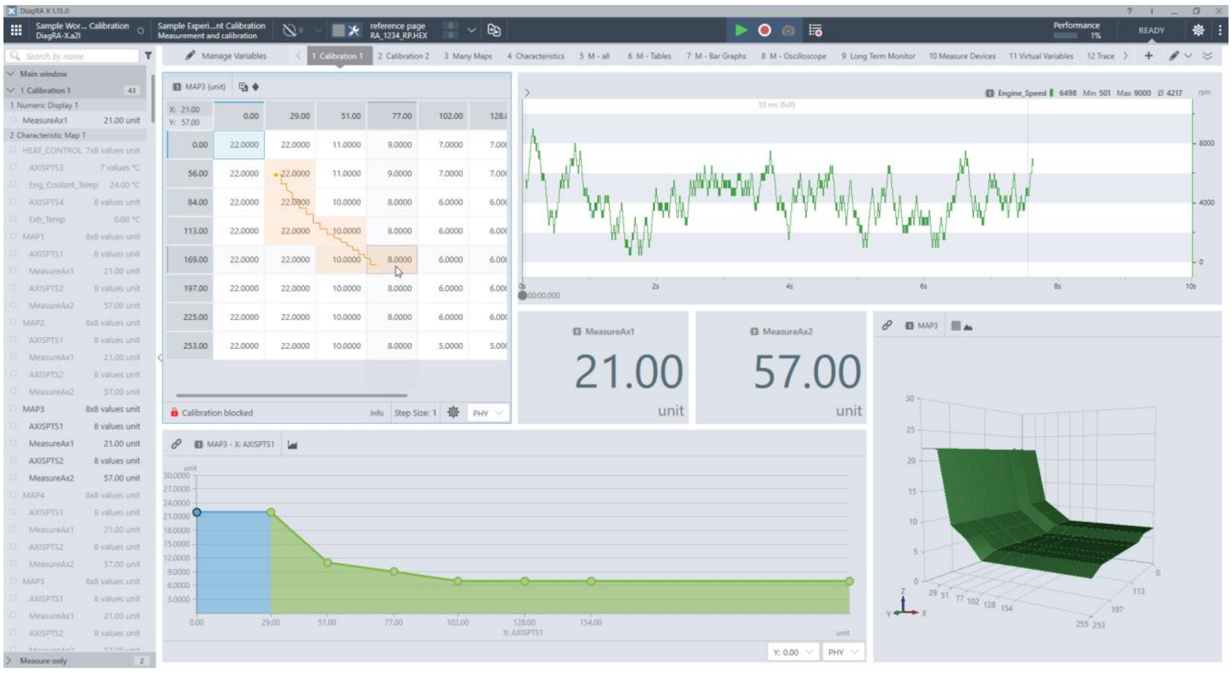


Figure 10: Worksheet overview



X: 21.00 Y: 57.00	0.00	29.00	51.00	77.00	102.00	128.
0.00	22.0000	22.0000	11.0000	9.0000	7.0000	7.00
56.00	22.0000	22.0000	11.0000	9.0000	7.0000	7.00
84.00	22.0000	22.0000	10.0000	8.0000	6.0000	6.00
113.00	22.0000	22.0000	10.0000	8.0000	6.0000	6.00
169.00	22.0000	22.0000	10.0000	8.0000	6.0000	6.00
197.00	22.0000	22.0000	10.0000	8.0000	6.0000	6.00
225.00	22.0000	22.0000	10.0000	8.0000	6.0000	6.00
253.00	22.0000	22.0000	10.0000	8.0000	5.0000	5.00

Figure 11: Calibration worksheet with some measurements for feedback.

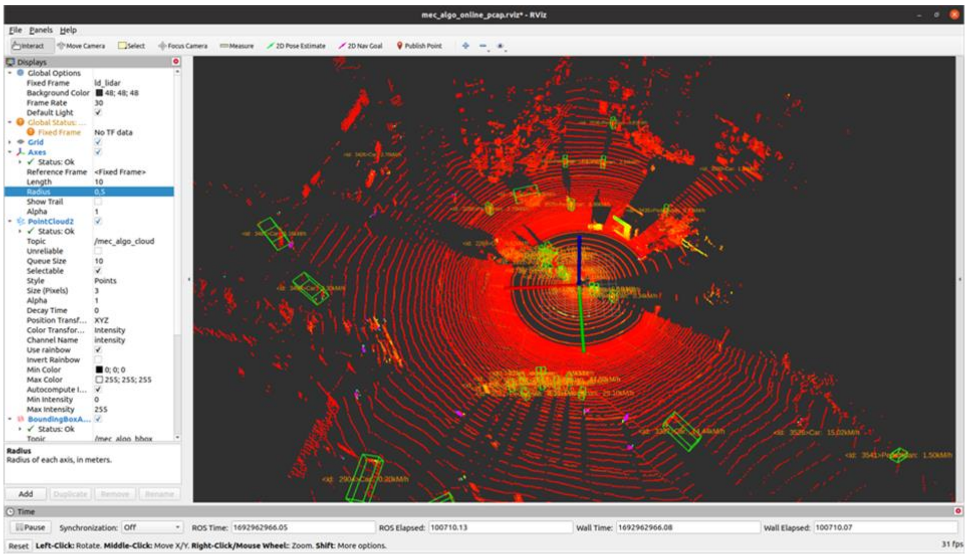
3.3.3 Recordings

For offline data analysis, measurement data can be recorded into the standardized Measurement Data Format, Version 4 (MDF-4). Within the experiment setup it can be specified which of the available labels (signals) are to be recorded when the recording is triggered. Additionally, there's a feature to record all labels of the current worksheet into a snapshot recording. A snapshot is just a dump of the measurements within the measurement buffer into an MDF-4 file. This is typically a sequence of a few seconds.

With the recording feature measurement data e.g., KPI-based measurements can be persisted for documentation issues, or for offline processing.

3.4 LiDAR Perception Software

For the validation of LiDAR sensors, a special hunter/target experiment was set up [12]. A LiangDao autonomous shuttle was used as target, for its software components see Section 3.5. The hunter was a LiangDao mobile station with a dedicated software for object perception. The software is ROS 1 (ROS: Robot Operating System [7]) based and contains a LiangDao specific AI based object perception algorithm. Due to the ROS background, raw data is published in *.bag files, object lists, which contain the information of identified objects (ID, position, speed…), are published in the same *.bag file and/or separate in *.csv files. A handover to external components is possible via ROS topics. These can be also sent via an MQTT module. The bag nature of the gathered data allows for a live streaming of raw and analysed data, and for streaming of a recording to other components to simulate a live setup.
For the visualisation of the data stream, RViz [13] can be used, a 3D visualizer for the ROS framework, see Figure 12 for an example.



Global Options	
Fixed Frame	ld_lidar
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
Global Status: ...	
Fixed Frame	No TF data
Grid	✓
Axes	✓
Status: Ok	
Reference Frame	<Fixed Frame>
Length	10
Radius	0,5
Show Trail	☐
Alpha	1
PointCloud2	✓
Status: Ok	
Topic	/mec_algo_cloud
Unreliable	☐
Queue Size	10
Selectable	✓
Style	Points
Size (Pixels)	3
Alpha	1
Decay Time	0
Position Transf...	XYZ
Color Transfor...	Intensity
Channel Name	intensity
Use rainbow	✓
Invert Rainbow	☐
Min Color	■ 0; 0; 0
Max Color	☐ 255; 255; 255
Autocompute I...	✓
Min Intensity	0
Max Intensity	255
BoundingBoxA...	✓
Status: Ok	
Topic	/mec_algo_bbox

Radius
Radius of each axis, in meters.

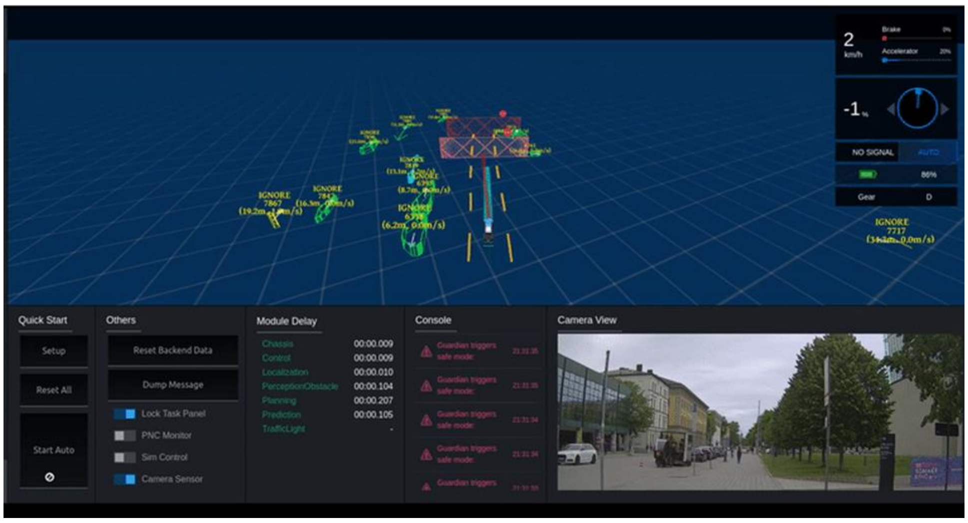
Synchronization: Off   ROS Time: 1692962966.05   ROS Elapsed: 100710.13   Wall Time: 1692962966.08   Wall Elapsed: 100710.07

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.

Figure 12: The LiangDao data steam for LiDAR object perception, visualized with Rviz. Red points represent raw LiDAR data. Green boxes mark objects identified and classified by object perception.

3.5 Autonomous Shuttle Operating System

The target in the LiDAR Validation experiment [12] was a LiangDao autonomous shuttle. The software responsible for control and data recording is Apollo ROS [14] based. The operating system itself, Apollo RTOS (RTOS: Real Time Operating System) is a modified Ubuntu Version with Apollo Kernel, the runtime framework is Apollo's operation environment. Apollo ROS is based on ROS [7] and Fast-RTPS [8]. The modified version of the shuttle provides a module-based GUI that allows to easily choose the components and functionalities needed for the given experiment, such as sensors, localisation, routing, perception or recording. See Figure 13 for an example. The operating system also allows a live view of the shuttle's status and perception during an experiment, see Figure 14. Due to the ROS nature of the system, data is provided via customized *.bag files (cyber record bag) and *.csv files and can be streamed via topics. Communication with ROS based systems can be established by the Apollo-ROS-bridge [15]. For the experiment, the software was used for shuttle control, and to identify its location for validation of LiDAR perception of the physical system.



Module Delay	
Chassis	00:00.009
Control	00:00.009
Localization	00:00.010
PerceptionObstacle	00:00.104
Planning	00:00.207
Prediction	00:00.105
TrafficLight	-

Figure 13: LiangDao shuttle status during an experiment. In the given example basic driving functions (Gear, Brake Accelerator, top right), a front camera view (bottom right), a map with detected objects (green), allowed maneuver area (orange stripes,) the planned path (blue line), the chosen path (red line) and designated stops (red, checkered planes) (middle) and status information (bottom left) are shown.

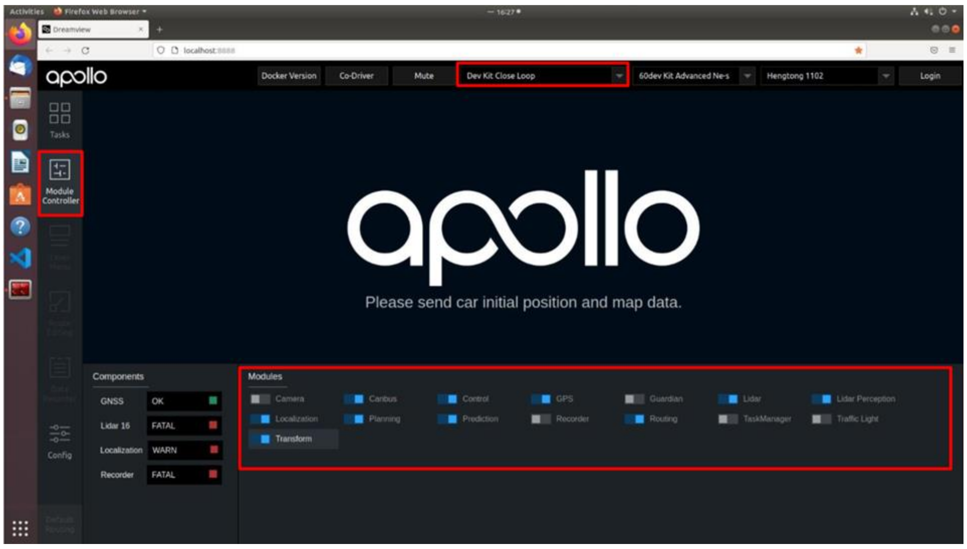


Figure 14: Apollo ROS based Module Controller of the LiangDao shuttle. The module switches allow to choose which components and functionality of the shuttle should be used for a certain experiment.

3.6 DaSense Platform

DaSense2020 is a modern AI-Platform that unites data management and data analytics.
The data management component provides an enriched view on files distributed across various data sources with an integrated full text search. This functionality enables users to quickly locate relevant information, even in scenarios where folder structures are not fully consistent.
Metadata can be added and extracted which facilitates the creation of parallel views on the data and organizing it tailored to the specific needs.

The analytics component allows to enrich the data by analyzing it via data science applications and to extend metadata or to generate reports.
These applications can be started directly within the data management platform on a selection of files.
The platform leverages large clusters for efficient big data analytics. In addition to using the applications, it also permits the development and deployment of new AI-applications within the platform.

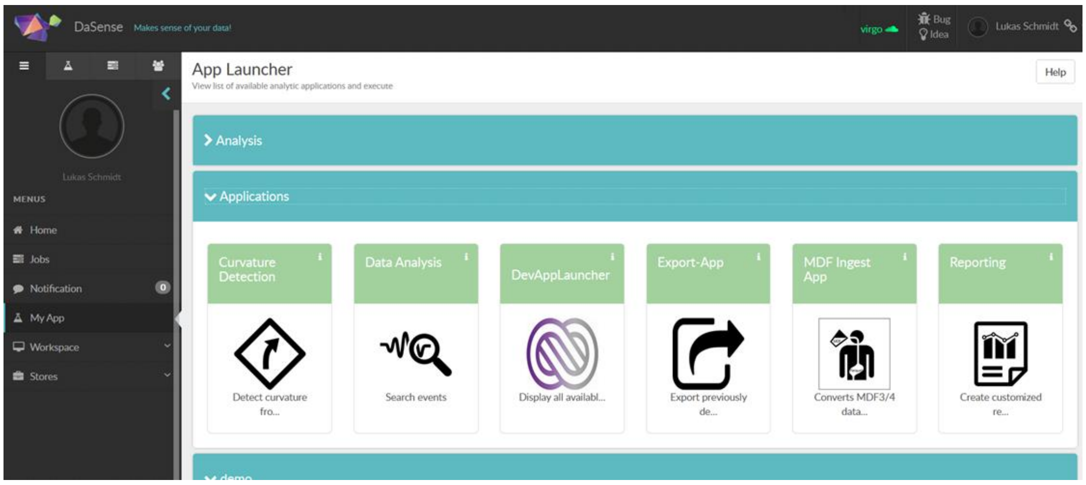


Figure 15: Data analysis platform with app overview

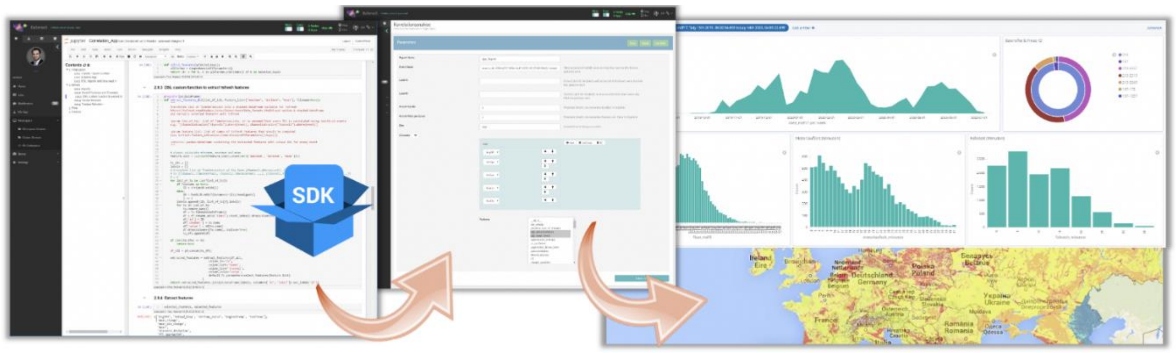


Figure 16: Development environment (left), compiled app with UI (middle), generated report (right)

In the UUV use case, DaSense can be used to provide convenient access to the ASIMOV solution through an AI-app, offering a graphical frontend for users and to initiate parallelized training and evaluation runs in a distributed environment. Data generated during these runs is then organized on the platform and available for subsequent analysis within the platform.
Notably, DaSense provides extended support for MDF-4 files that may be generated by the CPS. This includes the ability to search for the presence of certain channels and other MDF-4-specific metadata. Furthermore, histograms of channel-data can be easily computed and visualized via an integrated application. Additionally, an application facilitates data validation based on user-defined criteria, enabling automated validity and plausibility checks on extensive datasets.

3.7 Embedded SW Components

For some use-cases it might be necessary to have a more continues access to measurements or parameters within the target system. Therefore, within the ASIMOV project a set of embedded components has been developed to support the different tasks.

The tasks in focus are nearly the same as for the DiagRA-X tool:

- Measurements for the physical system as a basis for the model generation
  - Online measurements
  - Offline measurements (recordings)
- KPI-based measurements to validate the model quality.
- Tuning parameters within the physical system to force a specific behaviour / enable a special functionality.
- Writing of trained parameter-sets into the target system to update (AI-)components.
- Support KPI-based measurements to support metrics-based evaluation of the parameter set.
  - Online measurements
  - Offline measurements

Additionally further tasks are supported:

- Providing AI-based functionality attached to the physical system.
- Local recording capabilities while driving validation via an integrated embedded system.
- Direct integration of some interfaces into the http-based webservice structure of the overall simulation.

Additionally, a component has been added that allows to parametrize and execute an AI-Model within the embedded software using Tensorflow Lite. This component can be used to integrate some AI-based functionality into the target system, even if there’s not any environment for that already present. The component allows to configure the model structure used within the AI function, as well as setting or updating the trained parameters within the model to modify the functional behaviour of the AI-function.

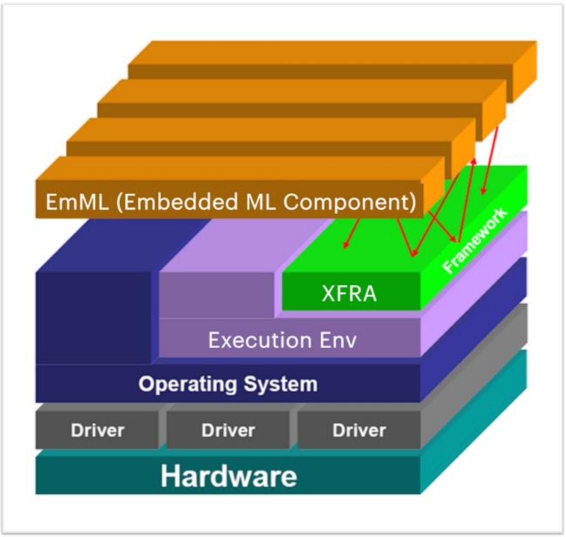


Figure 17: Structure of the XFRA Framework to support service-based components similar to OSGi in Java [16]

To have the whole embedded SW in a modular shape, a framework has been adapted inspired by OSGi, but fully implemented in C++ to achieve the best performance on even low performance hardware. OSGi is a Java-based modular framework that is used e.g., by the eclipse project.

The modular C++-based framework XFRA (eXtensible Framework by RA) provides the basic concepts of OSGi for C++ software:

- Bundles
- Services
- Observers
- Events
- Manifest information
- Class loading

Those services, that are used very often are already bundled with the framework itself. The basic services are:

- Logging
- HttpService
- HttpClient
- Interactive console

A common logging mechanism allows all bundles to write into a centralized log service. With that all log messages are written into one central file. As other bundles, alos the logging bundle is optional and could be skipped if there’s no requirement for logging.
The interactive console allows to inspect the system and execute basic functional tests by using supported commands the different bundles provide. Thus e.g., a database connection could be validated. As an optional bundle, the console could be skipped for a productive environment. But it could dynamically add to the running system, if needed for detailed inspection.
The HTTP-bundles are described further down in chapter 3.7.3.6

Besides these core features, the SW also supports:

- JSON data handling (parser, exporter, data model)
- INI data handling (parser, exporter, data model)
- ZIP support (zip/unzip files/archives)
- Support of CAN-interfaces to send and receive CAN frames.
- Support of the XCP protocol to interact directly with ECUs regarding measurements and calibration.
- Support of the MDF-4 file format for recording measurement data.

- A ring-buffer component to allow efficient data frame handling i.e., minimizing number of internal data copies.
- Configuration management to properly configure the modular system depending on the respective modules.
- Support of the respective description formats (importers for A2L, DBC) to configure the measuring components.

The framework and all the moules are written in a cross platform compatible way. Thus, the components are available for the following systems:

- Windows-PC x86 + x64
- Linux-PC x86 + x64
- ARM-based platforms (arm32, aarch64),
    - e.g., the FleaBox from CarMedialab GmbH
    - RaspberryPi based hardware.
- Power-PC based hardware

3.7.1 Bundle concept

To keep the system modular, the approach of OSGi has been followed in a way that the architecture supports bundles as functional containers that can be loaded and activated during runtime. In contrast to Java, where a bundle is basically a jar file and thus a zip-archive, in XFRA the bundle is realized as a dynamic library i.e., Dynamic Link Library (DLL) in Windows and a Shared Object file (SO) in Linux, respectively.

Like OSGi bundles an XFRA-bundle also provides manifest information and provides an Implementation of the BundleActivator interface to manage the activation and deactivation of the bundle. Typically, on activation a bundle registers services that can be used from within other services.
E.g., the JSON handling service is used by the HTTP-service to allow the implementation of REST based APIs.

The startup configuration specifies which bundles are to be loaded on startup of the framework.
Within the “[Bundles]”-section the bundles to be loaded are specified. For each bundle is defined, if it should be started, and the start level specifies when the bundle can be activated. The following configuration shows a sample configuration.

```
# ========================================
#        XFRA configuration file
# ========================================
# XFRA Framework V0.0.2.-1
# -- Friday 2023-01-27 16:46:40(13500) --

[Bundles]
xfra.logging = start
xfra.logging.file = start,1
xfra.console = start,2
de.rac.data.json = start,2
xfra.util = start,3
xfra.http = start,3
xfra.http.console = start,3
xfra.http.api = start,3

[xfra.console]
prepare-script = prepare.xcs
run-script = run.xcs
```

3.7.2 Service concept

The service concept allows each bundle to provide and/or consume services that are registered with the framework. For the management of the services in the overall system the framework provides the ServiceRegistry, a component that manages the service registrations, supports to query for particular services and provides an observer pattern to register event listeners that get informed on new services or services that are about to be removed.

Services are provided by bundles in different flavours. There are static services that are available as long as the respective bundle is loaded. As they are provided by a single instance accessible for all consumers, they are singleton services. In contrast it's also possible instead of a fixed instance to register a ServiceFactory. When a service consumer asks for a service, the ServiceFactory creates a new instance per request so that every service consumer has its own instance of the respective service. This is used, e.g., for data models (INI, JSON).

3.7.3 Service bundles available within ASIMOV

3.7.3.1 CAN-interfaces

Automotive components are very often connected via the Controller Area Network (CAN) bus. Thus, it's imortant to support that kind of interface to establish a communication with the car or its components. Even the OBD-Port provides predefined pins for CAN communication.

The architecture of the component is designed to support multiple CAN channels at a time. With this, it's possible to realize measurements e.g., with a CAN bus of the car combined with an additional CAN bus used for external measurement equipment.
Also, our architecture allows to reuse software components on different platforms. This is done with a common abstraction of CAN-interfaces realized for e.g., the SocketCAN [17] on Linux the same way as native drivers on Windows systems, either standardized interfaces like the SAE-J2534 (PassThru) [18], or the RP1210 devices or dedicated drivers for some interfaces as e.g., interfaces from Peak, Kvaser, Interpid Control Systems, Vector Informatics, ETAS, and others.

To support multiple platforms, several CAN interfaces are supported:

- SAE-J2534 (PassThru) interfaces (Windows)
- Dedicated drivers e.g., Kvaser (Windows)
- SocketCAN (Linux)

For integration with the other services, a generic class-interface for the CAN support is used. This allows to provide CAN support as a platform independent service to other bundles.

3.7.3.2 XCP support

To allow measurements and stimulations with ECUs in the automotive context the XCP [5] protocol is used. The 'X' within XCP represents the variable part of that standard: The transport layer. XCP can use several transport technologies while running the same protocol stack towards the application. So, within a car, CAN, Ethernet or FlexRay are used very often as a transport layer.
XCP allows target triggered data acquisition (DAQ) without the need to poll the information from the target device. Within our software, XCP is supported on CAN and Ethernet/IP (TCP + UDP). The XCP implementation for Ethernet/IP is also used within the ROS-XCP-Bridge (s. 3.7.3.3).

3.7.3.3 XCP-ROS Bridge

To gather information out of a ROS based system, the respective ROS-topic update events must be evaluated. This is done via a topic subscription within ROS. By that, the subscriber gets triggered as soon as new information for a given topic is available.

The XCP-ROS-Bridge allows to sample topic data on an update event and forward the information as an XCP-DAQ-Frame to an XCP-Tool, like DiagRA-X or the embedded XCP component. Thus, it's possible to retrieve ROS-data similar to automotive ECU-based measurements.
As XCP also supports the stimulation, i.e., injecting signal updates into the target system, the XCP-ROS-Bridge also allows to publish topics based on the XCP stimulation feature. When stimulation signals are sent to the XCP-ROS-Bridge, a predefined mapped topic in the ROS system will be published.

3.7.3.4 Measurement services

Common to measurements based on any kind of data frames (CAN, Ethernet) and even protocol frames, like XCP-DAQ-frames is the way of evaluating the physical measurement values.
For each of those data sources a description specifies the encoding of the signal's values. So, the first step in evaluating the raw frames is extracting the raw data bits from within the frame for each signal. After that the extracted bits are interpreted by the specified datatype of the raw value. Finally, this raw value is then converted into the final (physical) representation, i.e., the value could be scaled or mapped to some target data type. Only the physical representation allows clear semantic evaluation of the value, e.g., check for some temperature, or speed.

As the extraction of the signals value is a common functionality, it is provided by this separate component. It is used by the XCP- as well as the CAN-components to generate physical data frames that can be processed further.

Another common functionality based on measurement frames is trigger evaluation. Therefore, these functions are also part of the Measurement Services component. It enables the evaluation of trigger conditions based on certain signal values, which can be used to trigger further actions within the system, such as starting or stopping the recording of measurement data.

3.7.3.5 MDF-4 data format

Measurements should be stored in a re-usable data format. Therefore, the embedded components support the same format, that also DiagRA-X but also other common Measurement tools support: The ASAM Measurement Data Format, Version 4 (MDF-4). With DiagRA-X comes a separate measurement analysis tool DiagRA-X-Viewer, that also supports the ASAM MDF-4 format.

MDF-4 allows two modes of data capturing: ordered or unordered. The unordered way of storing is thought for embedded data loggers: They can write the data to the file as they receive it, without caring about any kind of order. The tools processing those files normally transform the unordered files into ordered files to allow more efficient analysis of the data.
The embedded component provided for ASIMOV produces already ordered MDF-4 files. With that the resulting files can directly be used for analysis within common Analysis tools.
To use storage resources efficiently, MDF supports compression of the recorded data. This is not yet supported but might be added in future versions of the bundle.

3.7.3.6 HTTP support

The pipeline setup within the ASIMOV process makes use of several webservice interfaces [12]. This is mainly used to couple the different components via specific flask implementations in Python.
To allow the seamless integration with those components, also the embedded components have been designed to interact via webservices in both directions.

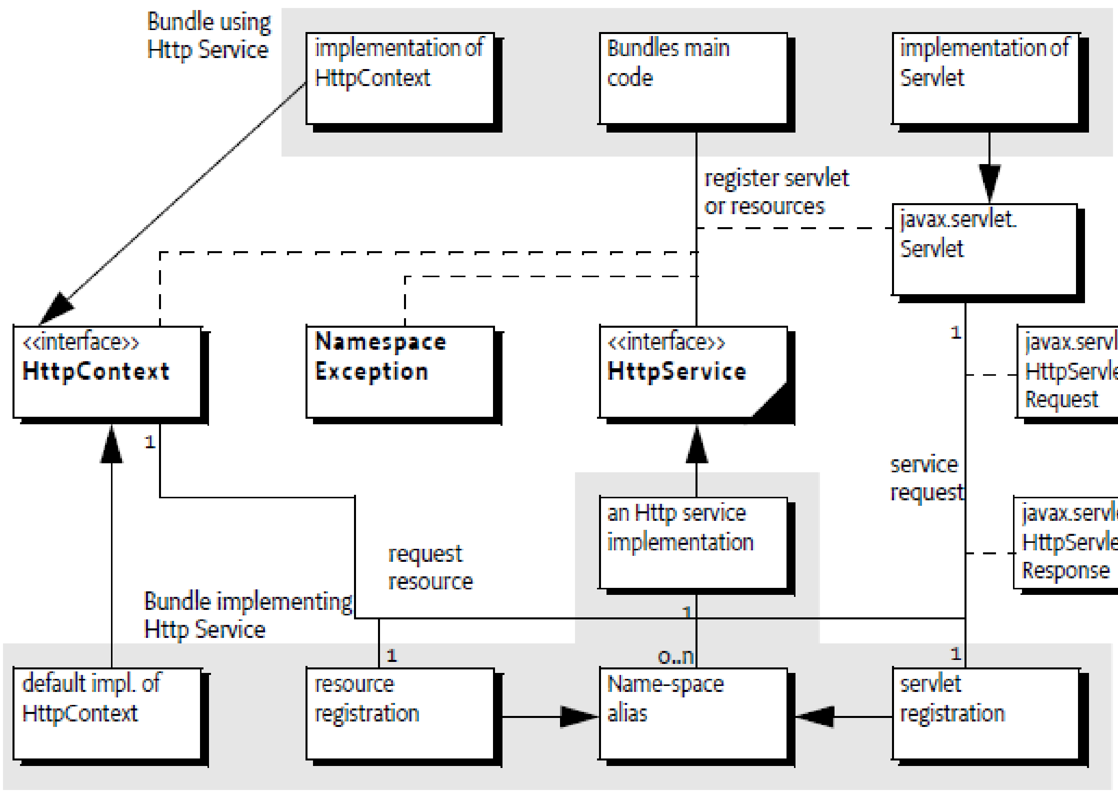

Bundle using Http Service
implementation of HttpContext
Bundles main code
implementation of Servlet
register servlet or resources
javax.servlet. Servlet
<<interface>> HttpContext
Namespace Exception
<<interface>> HttpService
javax.servlet.http HttpServlet Request
service request
javax.servlet.http HttpServlet Response
an Http service implementation
request resource
Bundle implementing Http Service
default impl. of HttpContext
resource registration
Name-space alias
servlet registration
1
0..n

Figure 18: Architecture of the OSGi HttpService [19]

To handle incoming requests, a HTTP-Service was realized. This service runs multiple HTTPServlets, each mapped to some sub-namespace within the servers URI. The architecture has been adapted from the OSGi-Framework [16] [19], but is implemented in C++ and thus run as native platform-specific binary. Similarly, the counterpart to read data from a webservice is the HTTPClient, that can connect to any kind of a HTTP server as e.g., a flask server.
With this component it is possible to seamlessly move the simulation environment towards the physical system. This might be necessary when the physical system should run in a partly simulated environment that allows to reproduce repeatedly testing situations as e.g., for the KPI measurements.

3.7.3.7 AI model support

As some physical systems might not already have an AI based implementation, or the access to the AI parameters might be restricted, a separate component has been adapted that allows to load and run the inference for a given AI-model.

The component is based on the TensorFlow Lite framework which allows to define a model structure, in terms of

- Number of inputs
- Number and layout of internal layers
- Specification of activation functions (e.g., Sigmoid- or ReLU-function)

Besides that, it allows to load additional to the model layout the trained parameter set.
With this component it's possible to provide an AI-based functionality within the physical system with the possibility to switch parameter sets and thus test different model variants.

The AI bundle has been tested even in standard hardware environments, especially without GPU support. The model variants shown in Table 2 could run a full inference step within 1 to 10ms on a standard PC hardware. On a 32bit-Arm-based hardware running at 500MHz, an inference step could be done within a 100ms-cacle.

Table 2: AI Model variants successfully tested with the Embedded AI component

Inputs	**Layer 1**	**Layer 2**	**Layer 3**	**Outputs**	**Activation function**
7	512	256	128	1	Sigmoid
7	128	512	256	1	Sigmoid
7	128	512	256	1	ReLU

Due to the technical design of the inference process, the inference time remains constant for the same model, when only changing the parameter-set, as the internal calculations are defined by the model structure. So once the model structure is validated to run in sync with the physical model, the parameter set can safely be improved without affecting the inference time.

With a 10ms response time, the inference fits into standard controlling loop times used in automotive ECUs. For further improvement GPU-support could be added using a respective hardware, as e.g., the NVIDIA Jetson Xavier AGX platform.

The AI component is intended to only do the inference of an already trained AI model. Thus, it provides the capabilities to load sucha a model from a file.

The AI model for the component is provided as a parameter file, and thus can be replaced easily for each evaluation run.

The TensorFlow Lite model is represented as a FlatBuffers [20] file using the "*.tflite"* file-extension.

4 Terms, Abbreviations and Definitions

Table 3 - Terms, Abbreviations and Definitions

CAN	**Controller Area Network**
CPS	**Cyber-Physical System**
DAQ	**Data AcQuisition**
DT	**Digital Twin**
KPI	**Key Performance Indicator**
PT	**Physical Twin**
KPI	**Key Performance Indicator**
ROS	**Robot Operating System**
XCP	**Universal Measurement and Calibration Protocol**

5 Bibliography

[1] ASIMOV-consortium, *"Methods and Tools for Training AI with Digital Twin - ASIMOV Deliverable D2.2",* 2022.

[2] ASIMOV-consortium, *"Architecture and technical approach for DT-supported AI-based system optimization - State-of-the-Art - ASIMOV Deliverable D3.3",* 2022.

[3] ASIMOV-consortium, *"ASIMOV Reference Architecture - ASIMOV Deliverable D4.a",* 2022.

[4] S. Aheleroff, X. Xu, R. Y. Zhong and Y. Lu, “Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model,” *Advanced Engineering Informatics,* vol. 47, 2021.

[5] Association for Standardization of Automation and Measuring Systems (ASAM), *ASAM MCD-1 (XCP) Universal Measurement and Calibration Protocol, Version 1.5.0,* ASAM e.V., 2017.

[6] C. C. R. B. B. J. Q. Charles M. Kozierok, Automotive Ethernet - The Definitive Guide, I. C. Systems, Ed., 2014.

[7] “ROS - Robot Operating System,” [Online]. Available: https://www.ros.org/. [Accessed 22 11 2023].

[8] EPROSIMA, “FastDDS - The most complete open source DDS middleware,” [Online]. Available: https://www.eprosima.com/index.php/products-all/eprosima-fast-dds. [Accessed 22 11 2023].

[9] AVL List GmbH, “Model.CONNECT(TM),” [Online]. Available: https://www.avl.com/de-de/simulationsloesungen/software-angebot/simulationswerkzeuge-z/modelconnect. [Accessed 22 11 2023].

[10] AVL List GmbH, “Simulation am Prüfstand mit Testbed.CONNECT(TM),” [Online]. Available: https://www.avl.com/de-de/entwicklungsgeschwindigkeit-methodik/verbindende-loesungen/simulation-pruefstand-testbedconnect. [Accessed 22 11 2023].

[11] RA Consulting GmbH, “DiagRA-X - die richtige Lösung für das Messen und Kalibrieren,” [Online]. Available: https://www.rac.de/unser-angebot/software/diagra-x/. [Accessed 22 11 2023].

[12] ASIMOV-consortium, *"Proof of Concept Demonstration and Evaluation of Unmanned Utility Vehicle - ASIMOV Deliverable D1.3",* 2022.

[13] “GitHub: RViz - A 3D visualizer for the Robot Operating System (ROS),” [Online]. Available: https://github.com/ros-visualization/rviz. [Accessed 22 11 2023].

[14] “Apollo Open Platform,” [Online]. Available: https://developer.apollo.auto/. [Accessed 22 11 2023].

[15] “GitHub: APOLLO-ROS Bridge,” [Online]. Available: https://github.com/ApolloAuto/apollo/tree/master/modules/bridge. [Accessed 22 11 2023].

[16] O. Alliance, *OSGi Service Platform - Core Specification, Release 4, Version 4.3,* The OSGi Alliance, April 2011.

[17] “GitHub: linux-can - Linux-CAN / SocketCAN user space applications,” [Online]. Available: https://github.com/linux-can/. [Accessed 22 11 2023].

[18] SAE International, *SAE J 2534-1_5,* SAE International, 2022.

[19] O. Alliance, *OSGi Service Platform - Service Compendium, Release 4, Version 4.2,* The OSGi Alliance, 2009.

[20] Google, “FlatBuffers,” [Online]. Available: https://flatbuffers.dev/. [Accessed 22 11 2023].

[21] ASIMOV-consortium, *ASIMOV - Full Project Proposal,* 2020.

[22] ASIMOV-consortium, *"Identification of relevant parameters modelled in DT - ASIMOV Deliverable",* 2022.

[23] ASIMOV-consortium, *"Application to Systems - Methods and Techniques - ASIMOV Deliverable D4.1",* 2022.

[24] W. M. v. d. Aalst, “Concurrency and Objects Matter! Disentangling the Fabric of Real Operational Processes to Create Digital Twins,” in *International Colloquium on Theoretical Aspects of Computing (ICTAC). Lecture Notes in Computer Science, vol. 12819*, Heidelberg, Springer, 2021.

[25] W. W. Group, *Web Services Architecture,* https://www.w3.org/TR/ws-arch/wsa.pdf: W3C, 2004.

[26] Association for Standardization of Automation and Measuring Systems (ASAM), *ASAM MDF - Measurement Data Format, Version 4.2.0,* ASAM e.V., 2019-09-30.