Association for standardisation of
automation and measuring systems

Ready, Set, Go!
Measuring, Mapping and Managing with XIL API 2.0

7th ASAM US-Workshop, Oct. 29, 2014, Novi, MI, USA

Dr. Rainer Rasche, dSPACE GmbH (Speaker)
Constantin Brückner, AUDI AG
Dr. Dietmar Neumerkel, Daimler AG

Preface

ASAM developed ASAM HIL API as a standard for the communication between test automation software and hardware-in-the-loop (HIL) test benches. HIL API enables users to choose products freely according to their requirements, independent of the vendor. Several implementations of the latest version of the standard, HIL API 1.0.2, are available on the market now. This is today. The future version 2.0 of the API is due out soon, with broadly extended functionality and enhanced applicability. It will support test benches at all stages of the function software development process –MIL, SIL, HIL, etc. As a result, the name “HIL API” is history. The ASAM standardization workgroup has decided to change the name to *XIL API –Generic Simulator Interface* with release version 2.0.

This paper shows some major benefits and use cases of XIL API 2.0 for *measuring*, *mapping* and *managing* by describing an example of distributed vehicle simulation in which variables from different data sources (real-time vehicle dynamics simulation model, virtual environment for road and other vehicles) need to be measured time-synchronously and concurrently. Measuring means collecting the time traces of these variables. Further test steps comprise data analysis and evaluation in order to provide verdicts within a test result.

The new *measuring* capability of XIL API 2.0 allows data acquisition to be configured. The time traces of variables from different data sources are assembled on a common time basis via ports already known from HIL API 1.0.2. Users can control the data flow using sophisticated triggers, e.g., with respect to important situations (gear shifts, transient response in the drivetrain, etc.). Recorders stream the coherent results either to memory for further processing or to standardized measurement files (MDF 4) for data reuse in a later process stage.

At the present time it is a challenging task for test developers to achieve effective decoupling between test cases and test benches in order to improve test reuse. The test developer’s work will be greatly facilitated by one of the new XIL API 2.0 framework’s basic features: *mapping.* This provides the standardized assignment of mapped values (variable identifiers, physical units, data types) on the test case and on the test bench side. Thus, mapped values can differ during the development process. The user just has to adjust the mapping –the test case (e.g., measurement definition) remains unaffected. In the example of distributed vehicle simulation, mapped values such as the data type or physical unit of the measured vehicle’s velocity may change on the test bench side because different precisions or different simulation model suppliers are involved in the development process while the same test case implementation is being reused.

Since port initialization is not part of HIL API today, methods for *managing* the ports’ lifecycles have also been added to XIL API 2.0. This enables the user to configure proper port preconditions in a unified way (e.g., which simulation model to download on the port and whether simulation needs to start automatically after download). This information enables the framework to put ports into their proper states before measuring begins.

All in all, this paper is intended to provide everyone working in test development and in tool development with some important use cases and best practices on what XIL API 2.0 is about: *managing* multiple ports as different data sources; *mapping* values; and *measuring* time traces of distributed data sources with products independent of vendor or process stage –all of which demonstrate the great advantages of this standard.

Contribution to the 7th ASAM US-Workshop, Oct. 29, 2014, Novi, MI, USA

Dr. Rainer Rasche, dSPACE GmbH (Speaker)

Constantin Brückner, AUDI AG

Dr. Dietmar Neumerkel, Daimler AG

Agenda

1 **Introduction**

2 Motivation using an Example

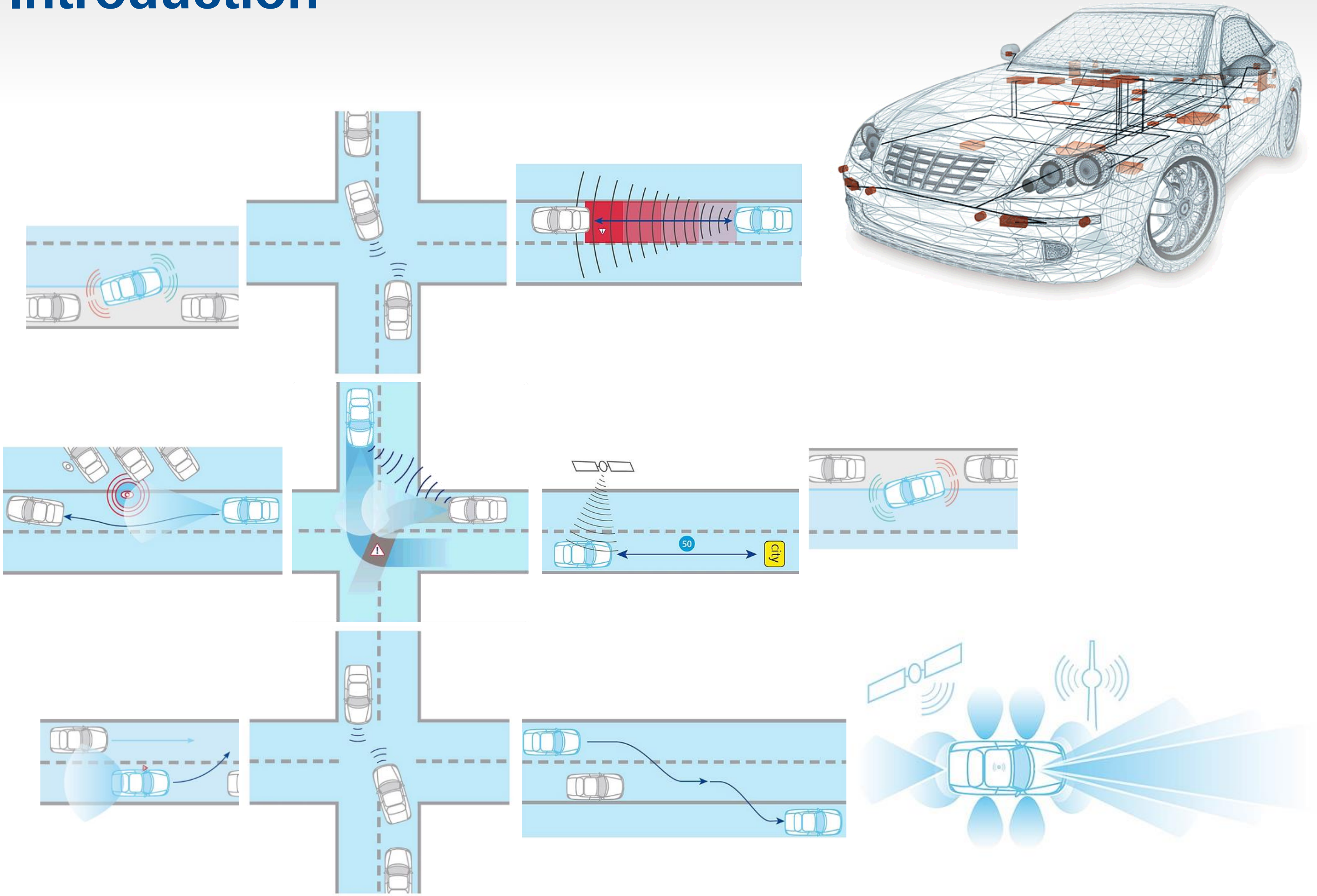
3 Measuring

4 Mapping

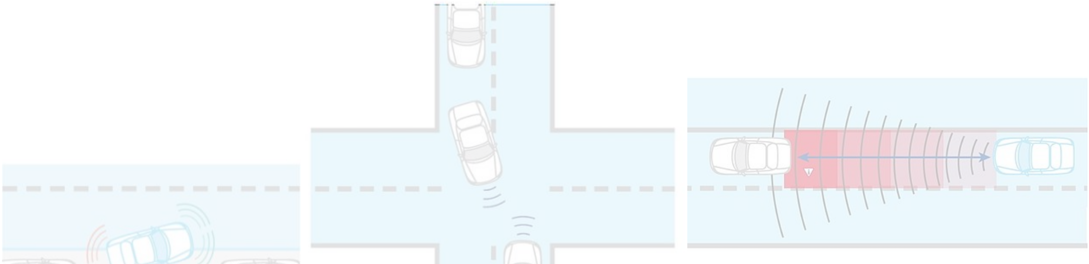
5 Managing

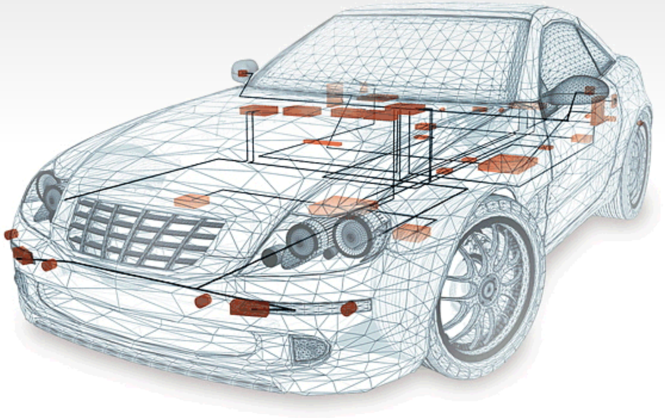
6 Summary and Conclusion

Introduction



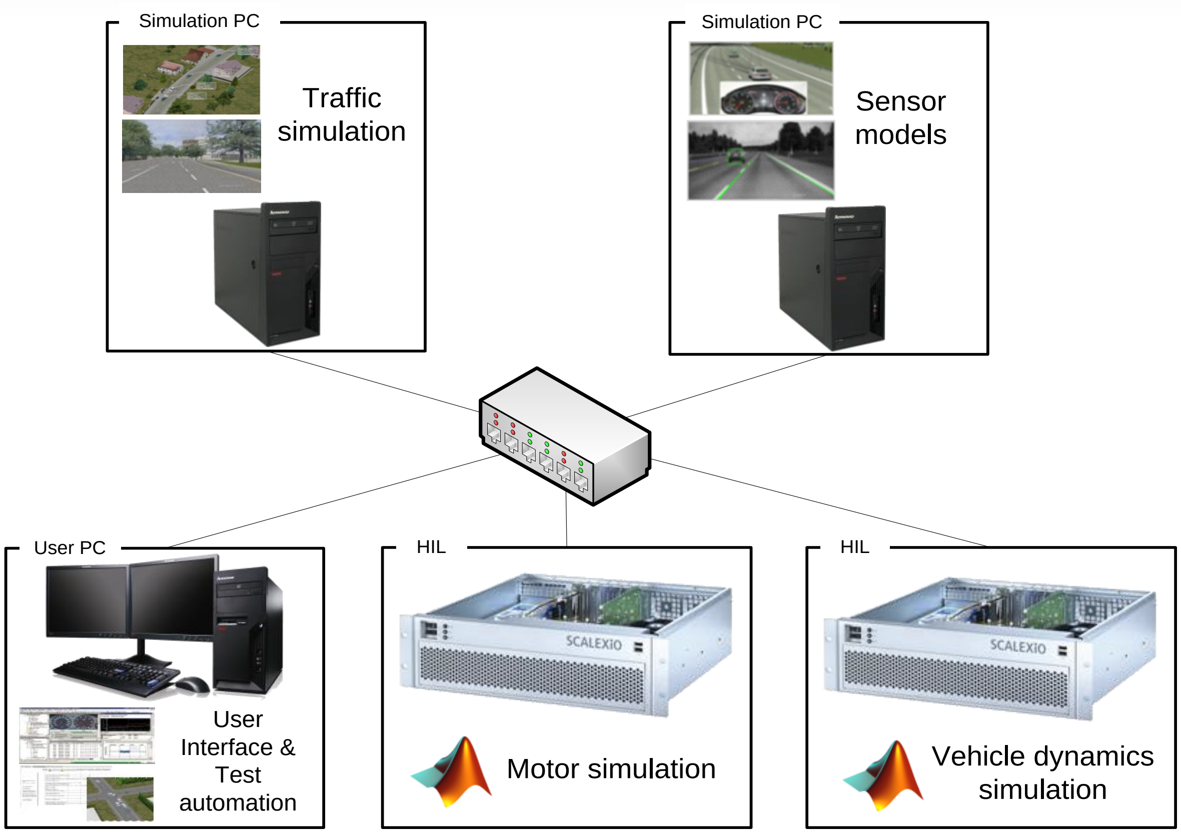
Introduction





- At present: Change from ECU-oriented to **function-oriented** development
- New functions: especially in advanced **driver assistance** or **autonomous driving**
- New trend: Functions are spread across a set of **collaborating ECUs**.
- Effect: **Increasing complexity** and difficulty in testing these functions
- Need: **Testing across ECU boundaries** for higher order functionalities
- Change: from homogenous single-purpose HILs towards a more **heterogeneous flexible architecture**
- Challenge: high **scalability**, **flexibility** and **reconfiguration** freedom to rapidly assemble different test configurations.
- **As a result:**
  Complex networked vehicles need complex networked test benches!

Virtual Environment



- Necessity to integrate new simulation systems for virtual environment
- Simulation of infrastructure like roads, buildings, landscape, traffic signs
- Enables Audi to test virtually in a complex interconnected surrounding world

Agenda

1 Introduction

2 Motivation using an Example

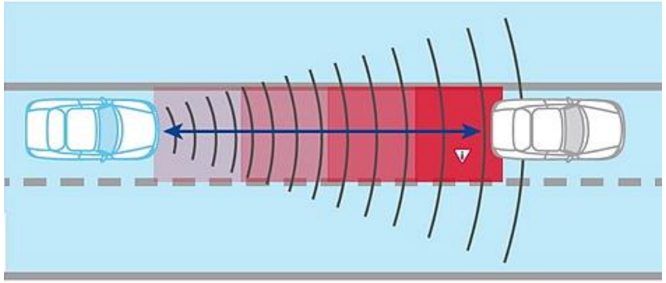
3 Measuring

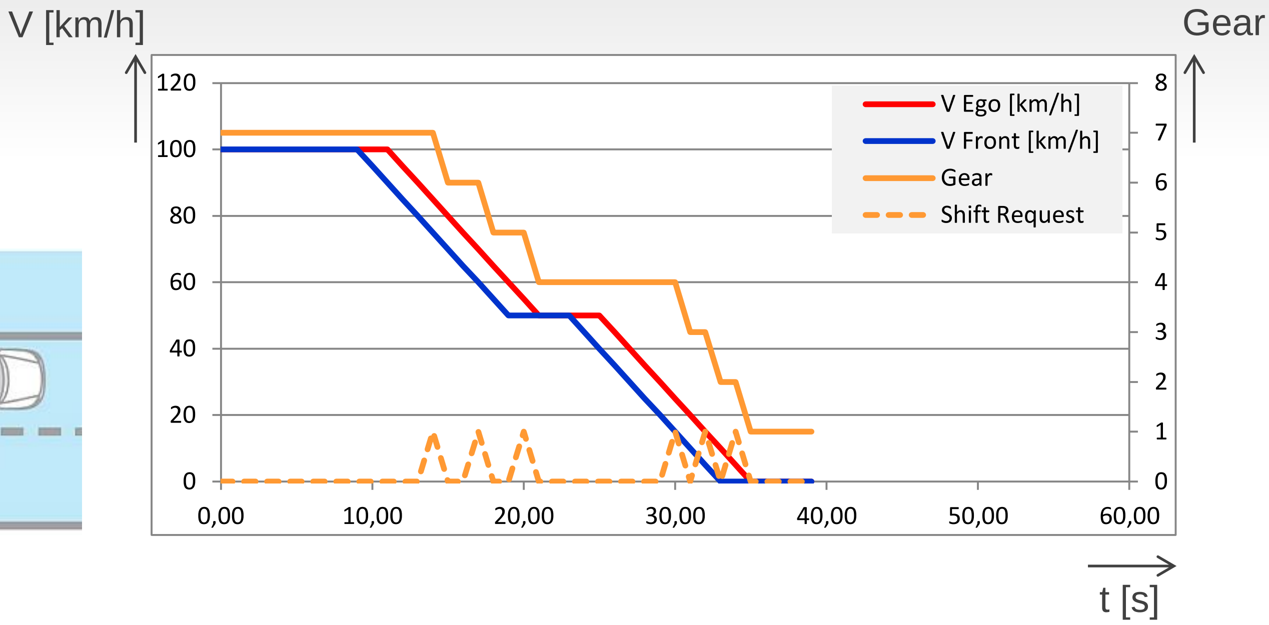
4 Mapping

5 Managing

6 Summary and Conclusion

Use Case





- A simulated car drives on a virtual road using adaptive cruise control
- Other traffic participants are simulated
- A sensor model simulates the radar sensor, which detects vehicles in front.
- If a vehicle is detected the velocity is adopted automatically to that of the car in front.
- The motor simulation adopts acceleration and speed of the car under test according to the received distance information.
- The vehicle dynamics simulation performs breaking if the car in front is getting slower. If necessary, the motor simulation sends a requests for a gear change to the transmission.

Agenda

1 Introduction

2 Motivation using an Example

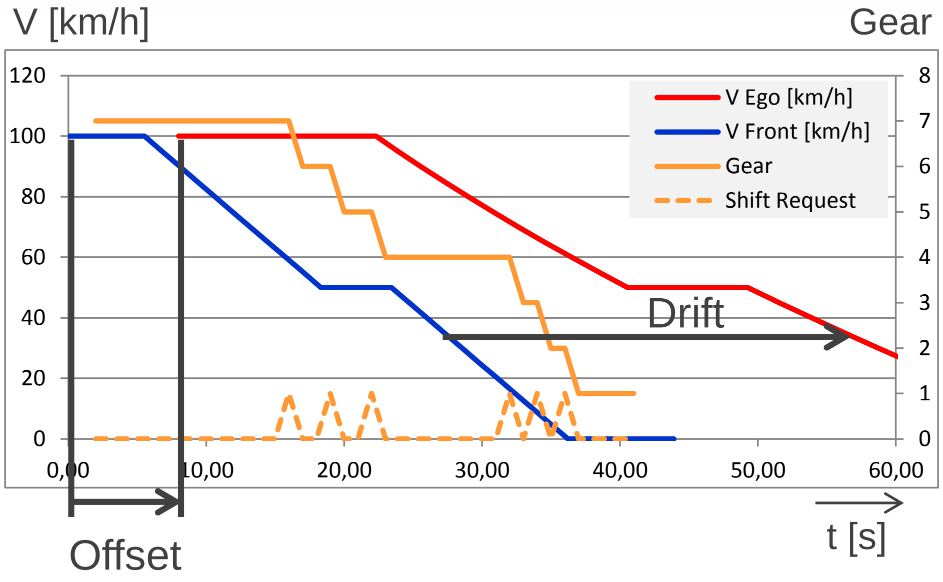
3 **Measuring**

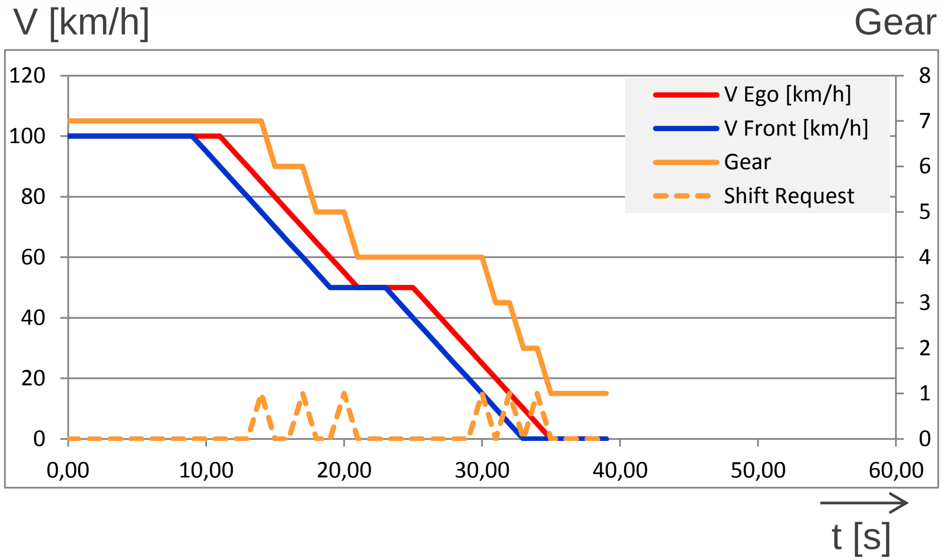
4 Mapping

5 Managing

6 Summary and Conclusion

Problem: Offset and Drift

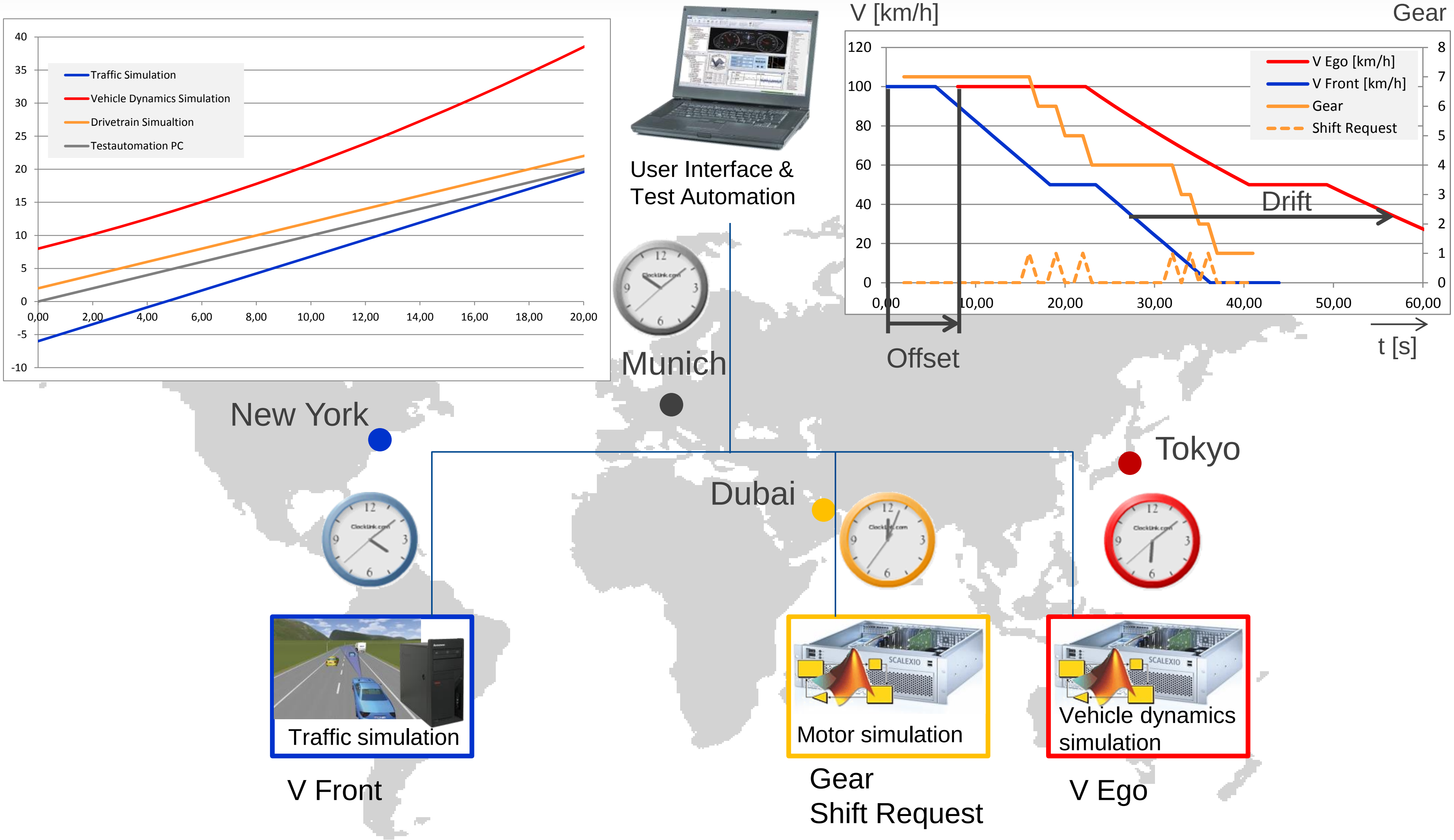




- Timers of different signal sources on different hardwares have offsets and drift apart due to different reset times and different frequencies which may also depend on environmental factors such as changing temperatures.
- XIL 2.0.0 provides interfaces to configure compensation of Offset and Drift on a common time base.

Distributed Simulation

Compare hardware timers with clocks at different places in the world



User Interface & Test Automation

Munich

New York

Tokyo

Dubai

Traffic simulation

Motor simulation

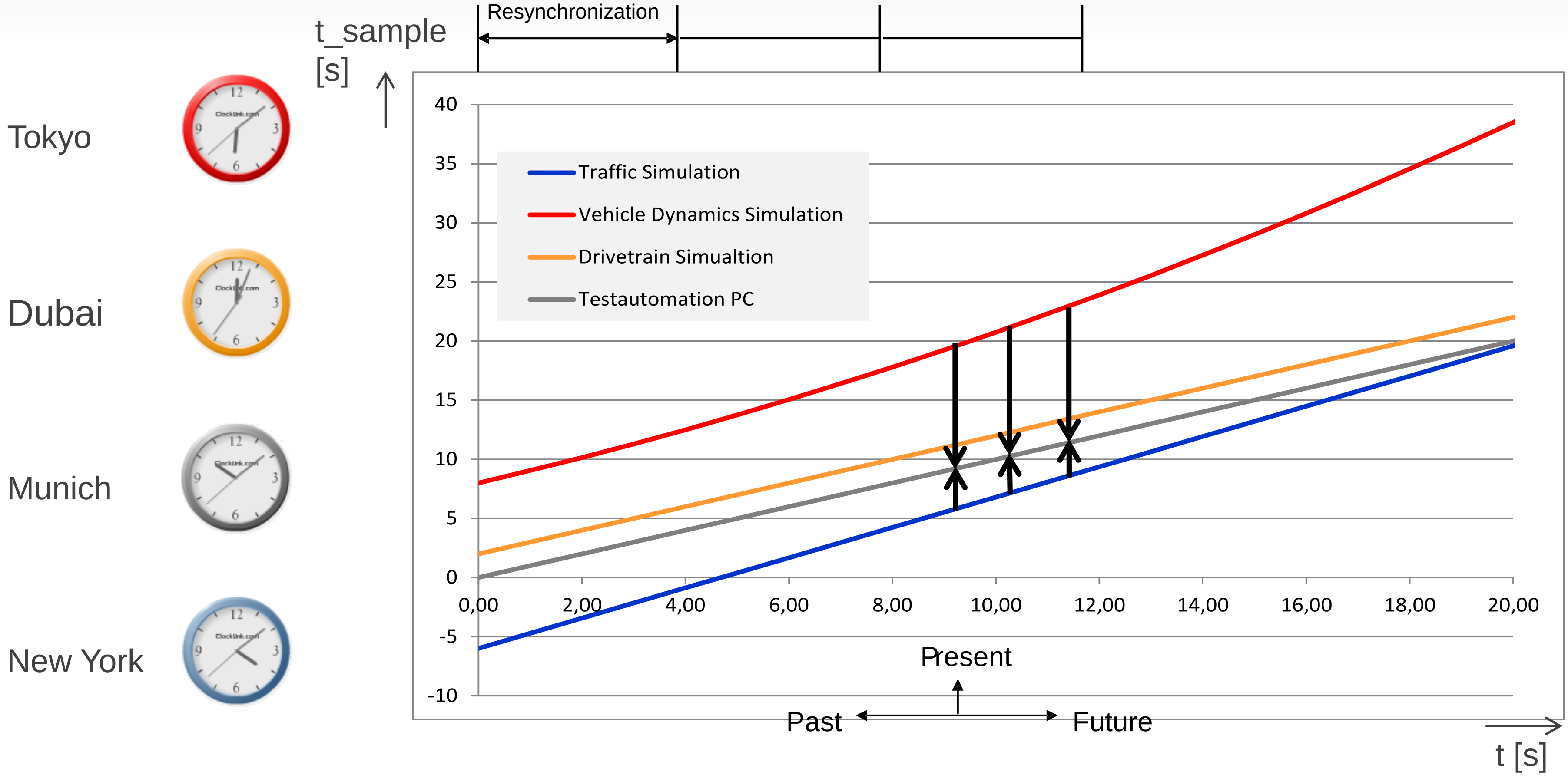
Vehicle dynamics simulation

V Front

Gear
Shift Request

V Ego

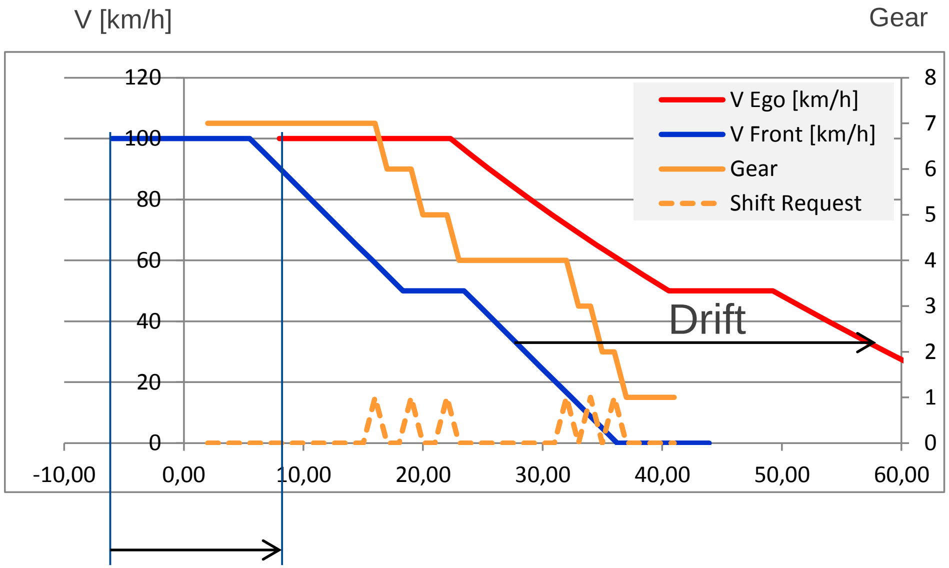
Shift of Time Stamps



- XIL 2.0.0 Framework allows to set an resynchronization interval in order to re-calculate correction factors for offset and drift for future data acquisition based on measured data history (past).
- The synchronized data, that XIL 2.0.0 Framework has added to the acquistion remains unchanged, which means there is no post precessing.
- The user can define recorders to store measured data either to memory or to file (MDF4 or higher)

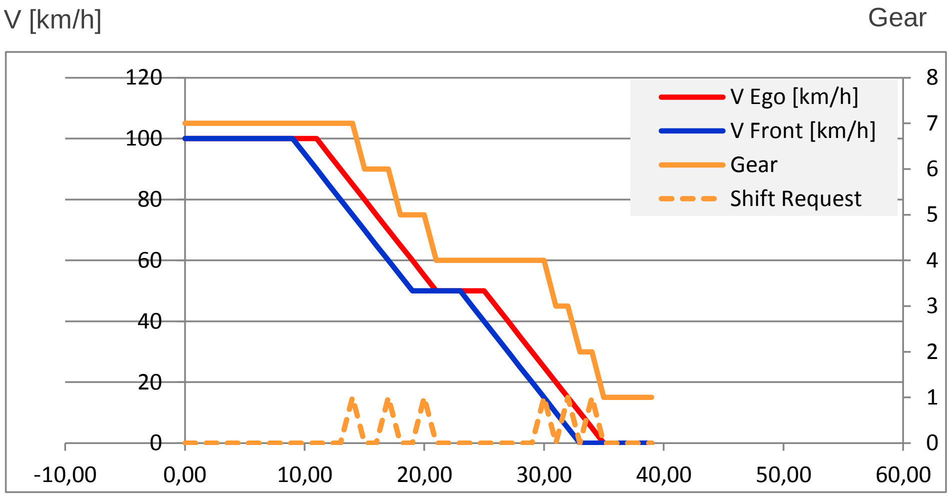
Result of Synchronization

Unsynchonized



Offset

Synchonized



Offset and Drift have been removed.

Agenda

1 Introduction

2 Motivation using an Example

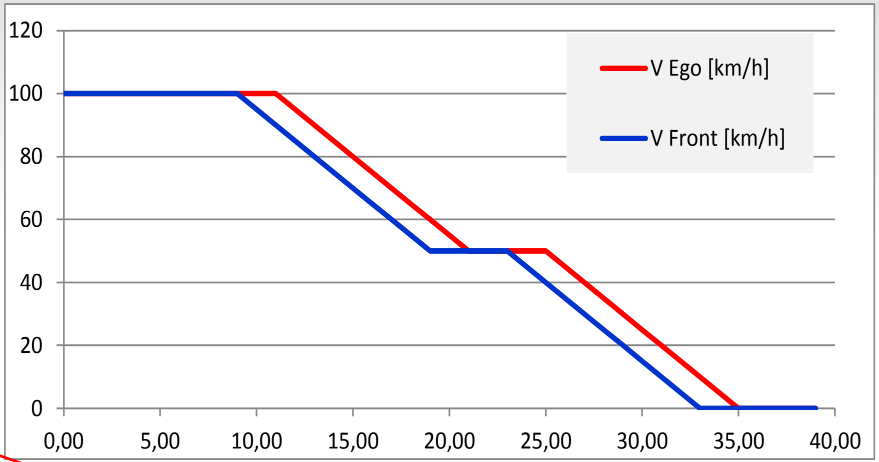
3 Measuring

4 **Mapping**

5 Managing

6 Summary and Conclusion

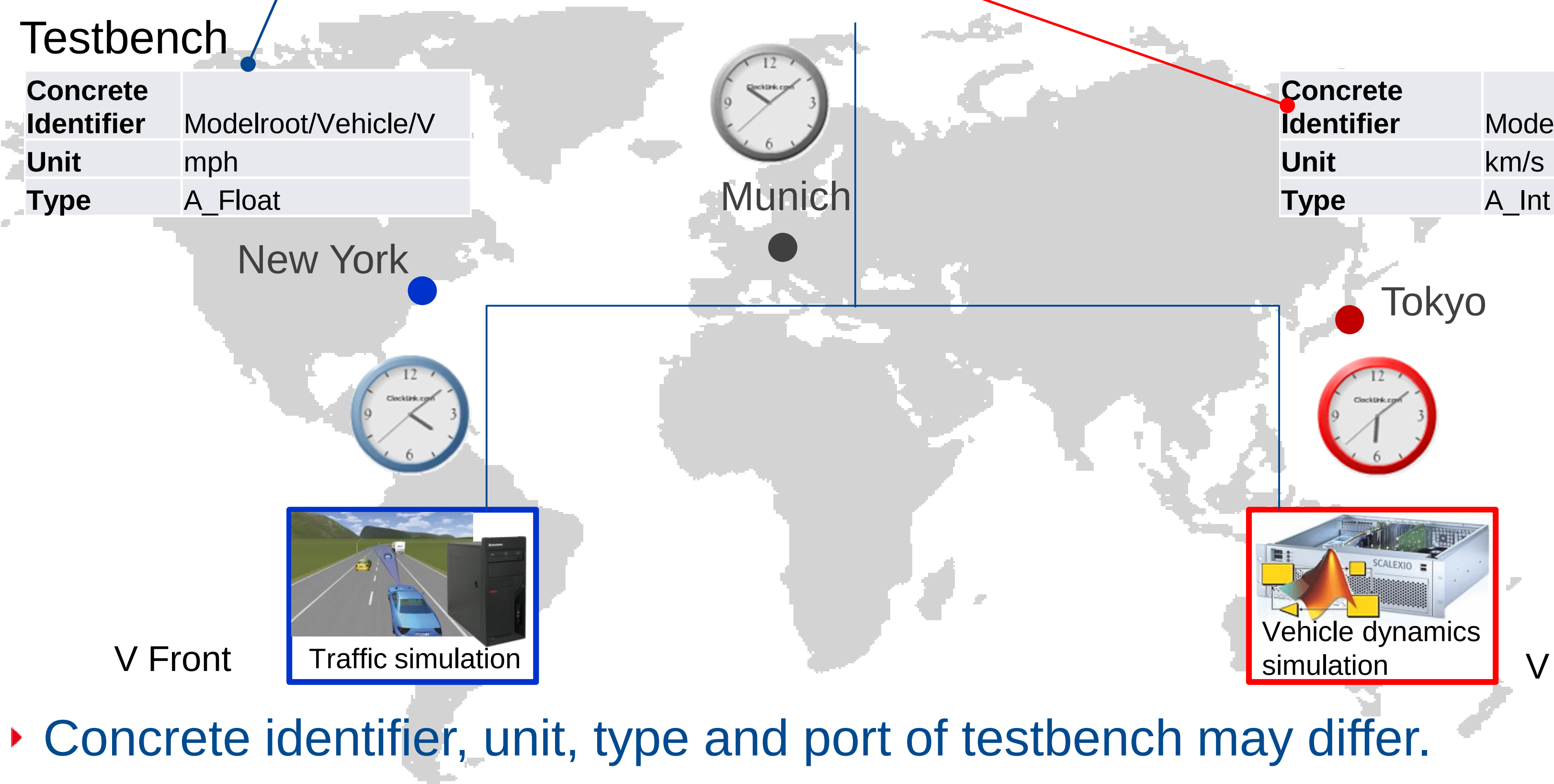
Mapping of Variables



Testcase

Abstract Identifier	V_Front	V_Ego
Unit	m/s	m/s
Type	A_Float	A_Float
Port	New York	Tokyo

User Interface & Test Automation



Testbench

Concrete Identifier	Modelroot/Vehicle/V
Unit	mph
Type	A_Float

Testbench

Concrete Identifier	Modelroot/VehDyn/V
Unit	km/s
Type	A_Int

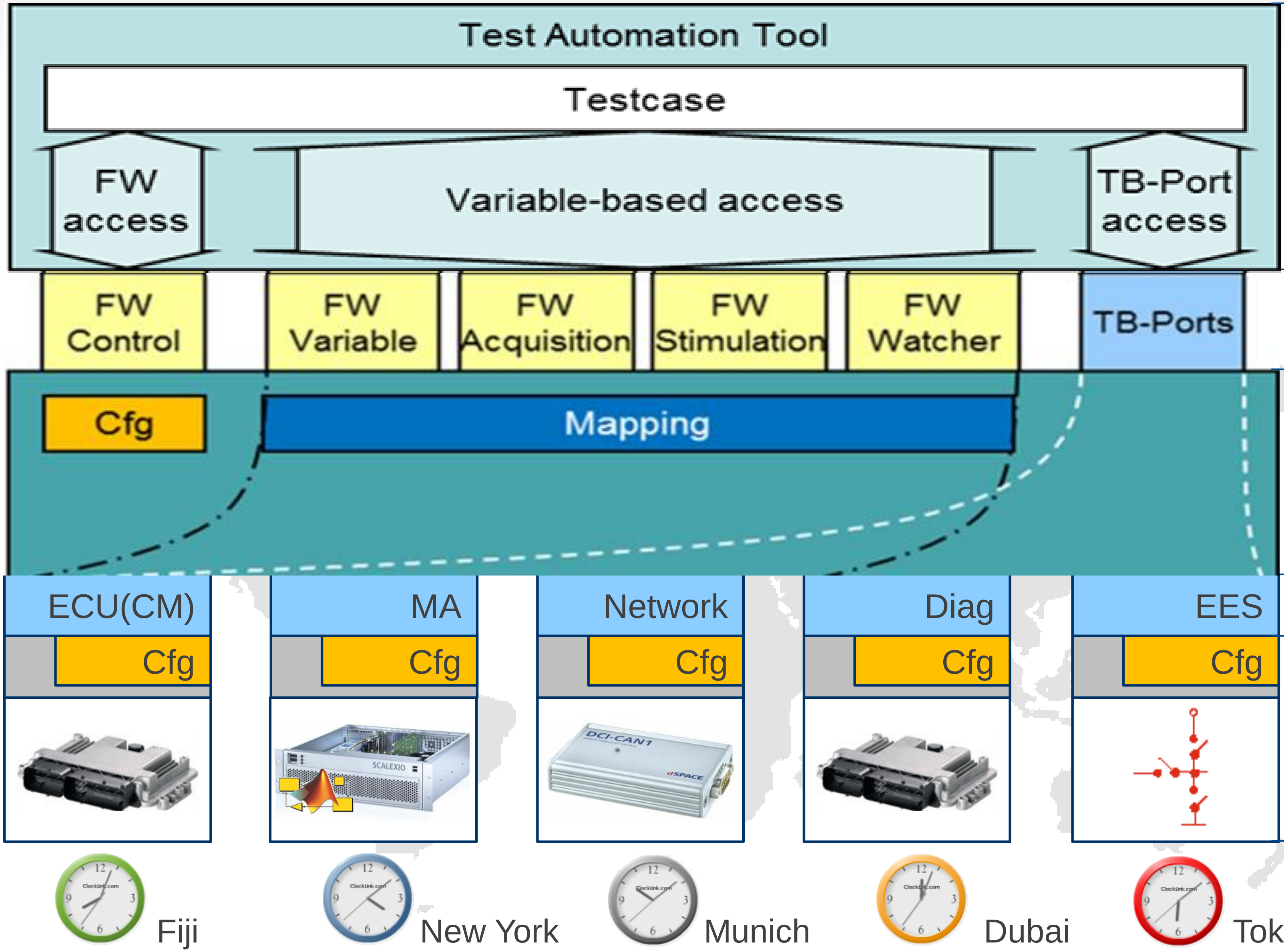
- Concrete identifier, unit, type and port of testbench may differ.
- The test case description remains the same!

Mapping

- **Motivation: Maximal Re-Use of Tests**

- **Identifier Mapping**
  Maps a framework label to a testbench label.
  E. g. maps an abstract identifier of a test case variable to a concrete identifier within the simulation model
- **String Mapping**
  Maps framework strings to test bench strings.
  For example this can be used to map abstract filenames on the framework side to concrete filenames on the test bench side
- **Raster mapping**
  Maps abstract raster names on the framework side to concrete raster names on the testbench side.
- **Based on XML Schema**

The Entire XIL Framework



Test Automation

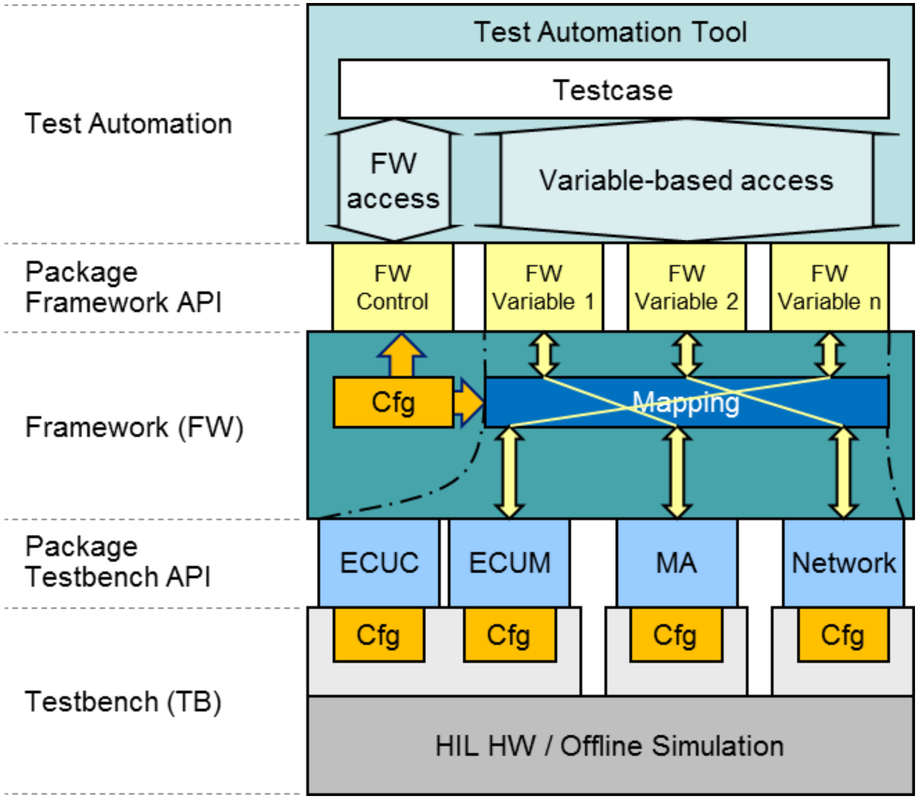
Framework API

Framework (FW)

Testbench API

Testbench (TB) incl. Tools/Drivers

Framework Variable



- Abstract Identifiers on test case side (e. g. “V_Ego”).
- Object oriented access
- Guarantees that test cases are independent of the underlying test system
- **Thus, independent of vendor and process stage ➔ XIL**

Agenda

1 Introduction

2 Motivation using an Example

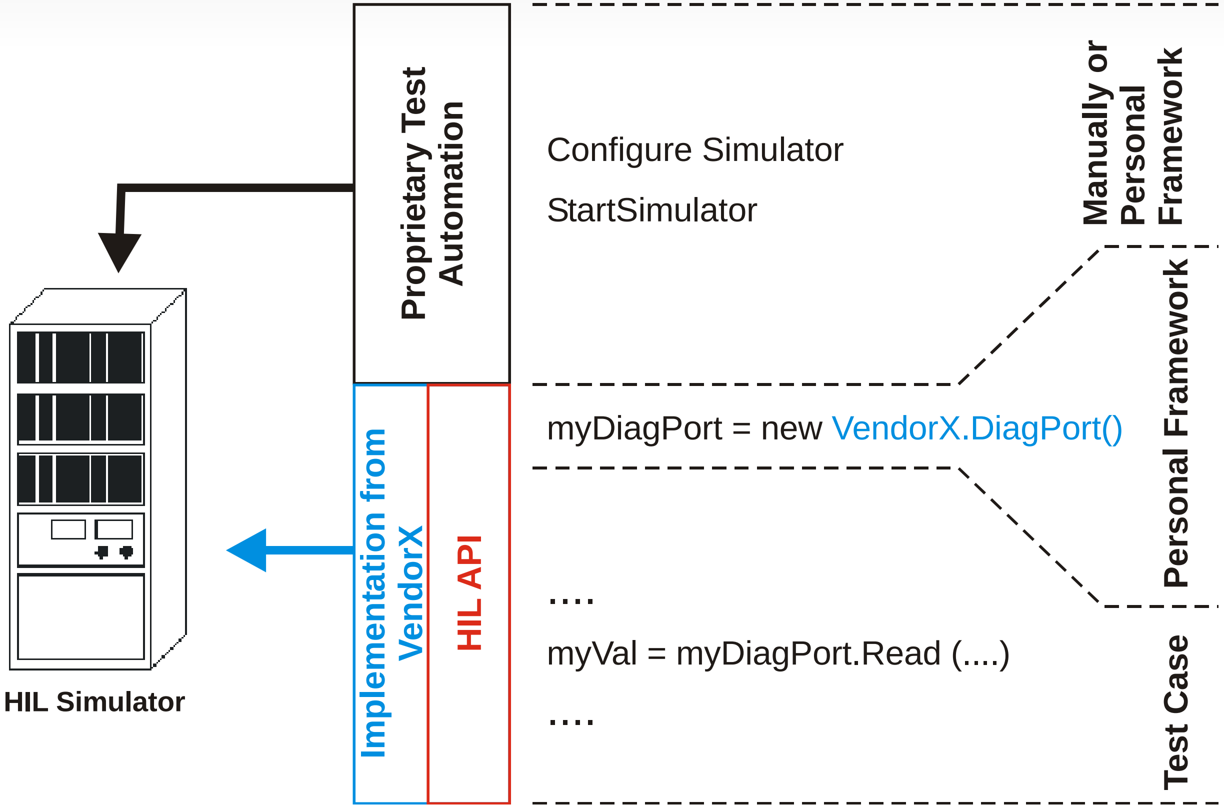
3 Measuring

4 Mapping

5 **Managing**

6 Summary and Conclusion

Initialization of the Simulator with HIL API 1.0.2


Proprietary Test Automation
Implementation from VendorX
HIL API
HIL Simulator
Manually or Personal Framework
Personal Framework
Test Case

Configure Simulator

StartSimulator

myDiagPort = new VendorX.DiagPort()

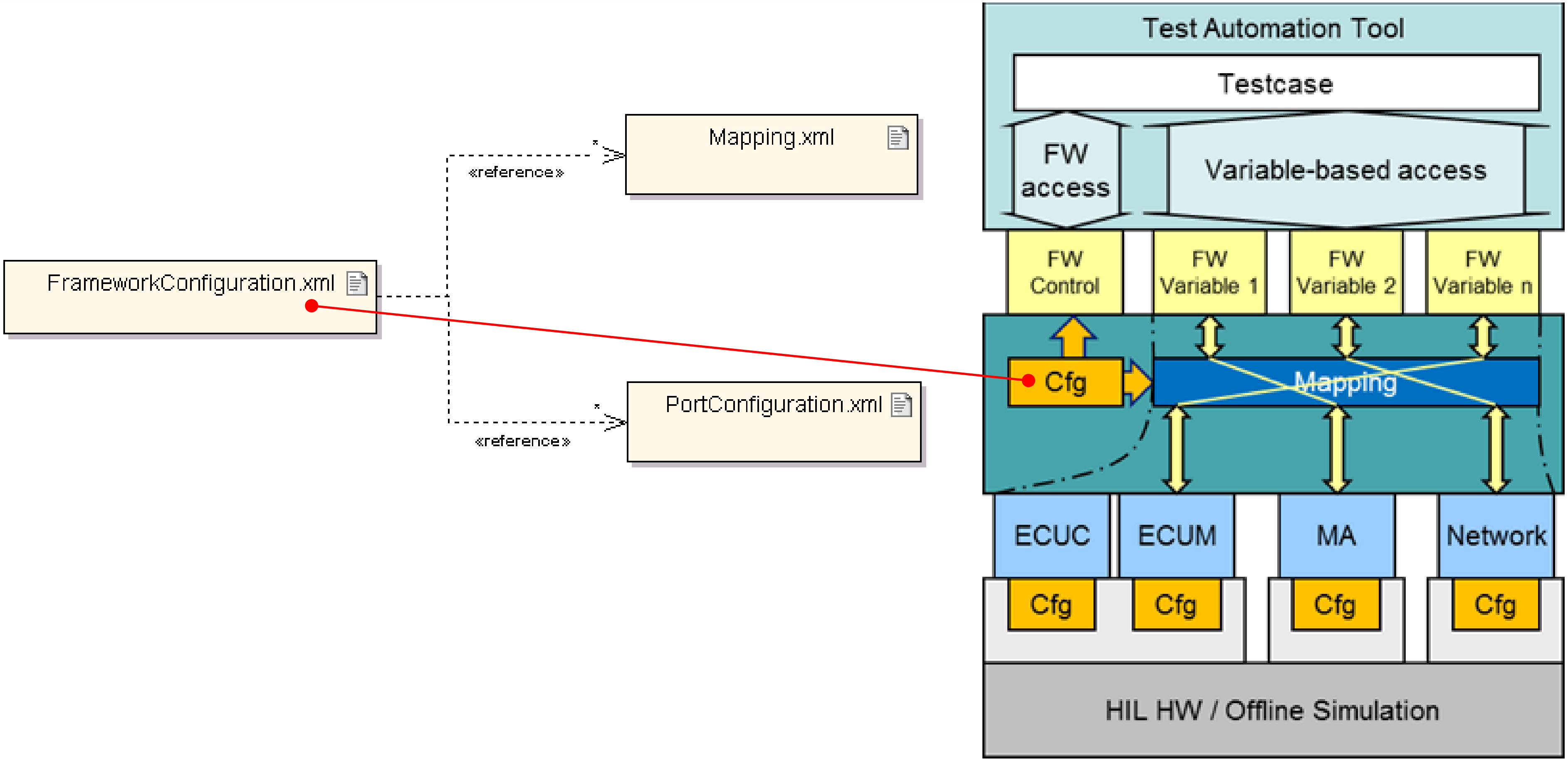
....

myVal = myDiagPort.Read (....)

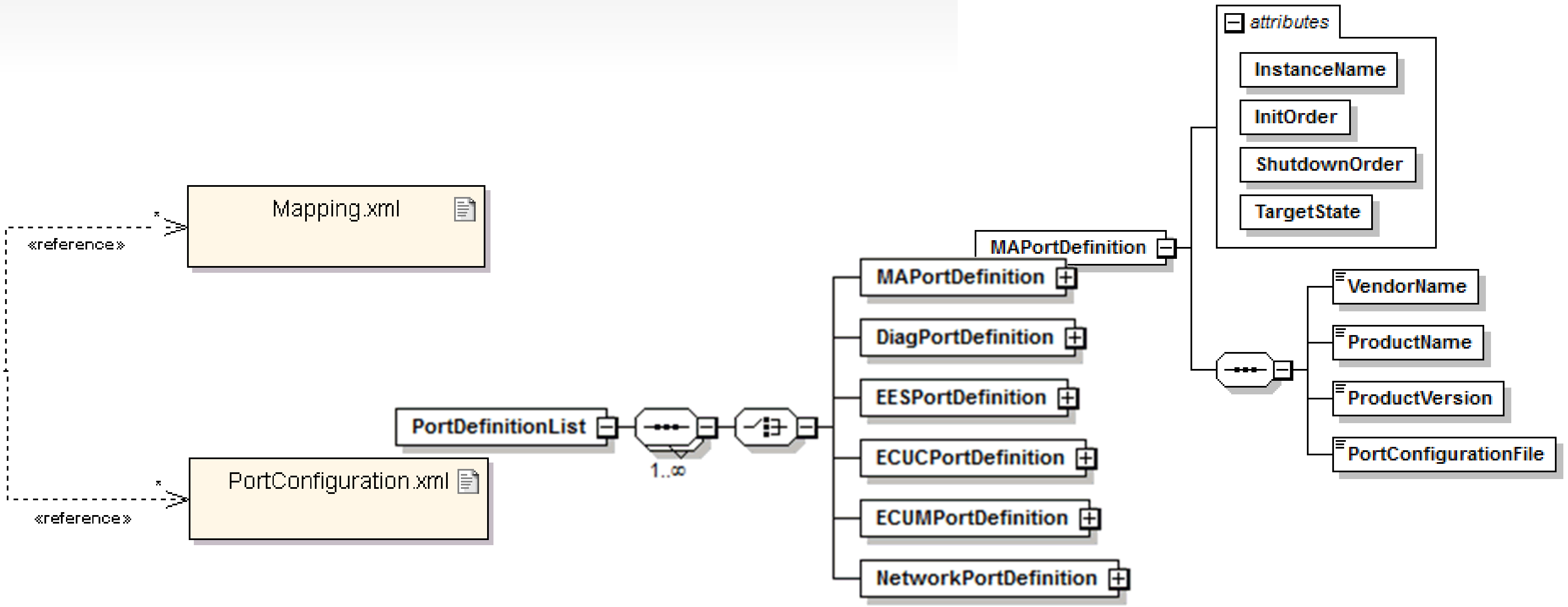
....

- No standardized methods available for initialization or configuration
- Thus, initialization was done manually or via personal framework

Framework Configuration with XIL API 2.0



Configuration of Life-Cycle Management



- Framework manages the port Life-Cycle
- Framework starts up ports in a configured order
- Framework establishes correct initial states (e. g. simulation stopped or running, online or offline, measurement stopped or running)
- Framework shuts down ports according to the shutdown order

Summary and Conclusion

- **XIL API 2.0** comes up with broadly extended functionality:
  Measuring of signals from different data sources with time synchronization
  Mapping in order to decouple test cases and test benches
  Managing of the test bench ports' life-cycle
- Easy test case exchange between different
  vendors and even between different development stages, e. g.
  offline simulators in early stages and productive HIL test benches
- Better know-how transfer from one test bench to the other
- Reduced training costs for employees
- **From end users perspective:**
  This allows the 'best' test software combined with the 'best' test hardware.