A safe and certified automated driving platform for different levels of automated driving

Eric Schmidt
Director Lead Engineering & Safety
TTTech Auto AG
We are at the intersection between the physical world and the cyber world.

SAFETY
REAL-TIME
SECURITY
ROBUSTNESS
RELIABILITY
FAIL-OPERATIONAL
Building synergies across safety-critical industries

SAFETY
REAL-TIME
SECURITY
ROBUSTNESS
RELIABILITY
FAIL-OPERATIONAL

Building strong integration platforms

Automotive
Aerospace & Space
Off-Highway
Industrial & IoT
The automotive industry is evolving at high speed

**YESTERDAY & TODAY**
- **Distributed E/E architecture**
  - n ECUs: n functions
  - (1 ECU: 1 function)

**2020 – 2025**
- **Domain E/E architecture**
  - 5-7 ECUs: n functions
  - grouped by functional domains

**2025 +**
- **Centralized E/E architecture**
  - 2 ECUs: n functions

Service-oriented architecture with real time guarantees
But how is the timeliness of the system achieved?

How is real-time behavior guaranteed?

This is quite „hidden“ in the design decisions of the architects.

Fundamentals of (smart/electronic) systems

**MotionWise**

Sense = See  Think = Compute  Act

Time-aware safety software platform

- Sensors
- ADAS/AD ECU
- Actuators
What is time-aware architecture?

The “actions“ of the system are majorly triggered by the progression of time (not by external events or interrupts).

Prerequisites needed to build a time-aware architecture:
- Planning (a system-wide schedule)
- Global time (time synchronization)

This applies to:
- The execution of software
- The sending and receiving of messages
The orchestra – an example for a complex time-aware architecture
Highly automated driving – based on a time-aware architecture

MotionWise

Real-time orchestration
- Time aware architecture
- Data synchronization layer
- Global scheduling

Safety by design
- Freedom from interference
- Error and health management
- Fail-operational requirements

Open integration platform
- Unified APIs
- Heterogeneous SoCs
- Platform services & tools

Processing

Deterministic Ethernet

PCIe
MotionWise platform software – handling the complexity

Deterministic Ethernet Backbone & PCIe Backbone

FlexRay

CAN

Ethernet

Ethernet

Autosar OS

LINUX

RTOS

LINUX

QM

ASIL D

ASIL B

ASIL D

ASIL B

ASIL B

Multi-Core MCU Safety Host

Multi-Core SoC Performance Host

Performance Host

Performance Host

Performance Host

Multi-Core SoC Performance Host
Motivation for a time-aware/time-triggered architecture

Imagine a regular Status Report Meeting

There are many people in a room. Each of them wants to speak. How can each of them present their status information?

- Decide on a fixed order of speaking
- Allocate 5 minutes to each of them
- Nobody is allowed to interrupt the others
- Schedule the next meetings (for the regular update of the status information) at a predefined time with a suitable cycle

Conclusion

(This applies to complex ECU as well)

- Make it time-triggered when you have regular (cyclic) activity
- Make it time-triggered when you have to coordinate many participants
- Make it time-triggered when you need a reliable and safe solution
- Make it time-triggered when you need intelligent cooperation
Advantages of a time-aware architecture

- **Deterministic behavior** → increased stability and testability
- Supports **complexity reduction** → improved designs, less test cases
- **No overload scenarios** → no messages lost, no extreme latencies
- Applications executed synchronously to communication → **stable real-time behavior** with constant duration of “computation chains”
- **Synchronized global time** → allows data exchange with global time stamps
- **Increased composability** → adding additional functions (in predefined time slots) does not require complete re-evaluation of system
- Allows **offline planning** and **optimization of the schedule** → delivers foreseeable timing behavior of the system and less “experimental surprises”
Scheduled data flow with guaranteed latency for computation chains
Typical distribution of latencies

A comparison of two implementations of an exemplary system

shorter “best case“ latency

uncertain “worst case“ latency

guaranteed and shorter “worst case“ latency + smaller variation
“Disadvantages” of a time-aware architecture

- The “best case reaction time” (best case latency) of an event-triggered system is not reached … but the worst-case latency is the critical parameter for safety

- Determining the relevant processing cycle times in advance and designing the system to be “only” this quick in its reaction times → designing in the maximum latency

- Defining/developing a schedule (for SW execution and communication) → “frontloading” the development process

- Flexible redistribution of computation resources (CPU execution time) is not supported (or more precisely … must be restricted to certain parts of the software) → critical parts of the system need to be designed for the worst-case conditions

- Synchronization between different time sources (e.g. for the software execution on different hosts) is needed to avoid additional latencies
How time-aware architecture accelerates the development
Conventional integration of complex real-time systems

01
Integration of platform without configuring execution frames.

02
Applications are integrated and tested individually by APP suppliers without any timing restrictions.

03
All applications are integrated by the SW-integrator on the platform; conflicts start immediately as it is not clear who is causing problems and why.

04
Conflicts are reported back to function SW suppliers, applications have to be modified to meet the system's timing restrictions.
Robust parallel integration process based on a time-aware architecture

01
Platform configuration includes execution boundaries for the applications.

Robustness through clear allocation and monitoring of resources (memory, CPU, comm.)

02
Applications are integrated and tested individually by the APP suppliers into their respective execution boundaries.

Parallel Integration to speed-up software development of multiple-software suppliers

03
All applications are integrated and are immediately able to run together; violations by APPs are detected easily.

Complete software integrated for functional testing
Robust parallel integration process based on a time-aware architecture

Robustness through clear allocation and monitoring of resources (memory, CPU, communication)

1. Configured platform with execution boundaries for the applications.

Parallel Integration to speed-up software development of multiple-software suppliers

2. Parallel integration and test of applications (by multiple suppliers) within their respective execution boundaries.

Complete software integrated for functional testing

3. All applications are integrated and are immediately able to run together. Violations of SWCs are detected easily.
Boundary Conditions

- ASIL D safety goals on system level must be supported for AD and ADAS functions (for systematic and random (HW) faults)
  … but … key constituents fulfill ASIL D just in parts (e.g. high-end SoCs, HW accelerators, BSPs, Libraries, legacy SWCs, …)

- SWCs of different ASILs + QM need to co-exist with freedom from interference (FFI)

- Checker-type monitors needed to detect dangerous faults and react

- Availability is becoming a safety property (transition from fail-silent to fail-operational)

Solution = Time-aware architecture

- gets complexity under control

- ensures deterministic real-time behavior (with well defined “wiggle room”)
Autonomous operation requires fail-operational systems

20+ years of excellence in designing, building and assuring fail-operational systems across safety-critical industries

Fail-silent

Off-highway
IEC 61508

Fail-operational

Aerospace
DO 178C / 254

Automotive
ISO 26262

Industrial
EN/ISO 13849

Autonomous operation requires fail-operational systems
From Fail-silent to Fail-operational systems

L0
Driver only

L1
Driver assistance

L2
Partial automation

L3
Conditional automation

L4
High automation

L5
Full automation

Active Safety
Pre-Crash Assistant
AEB

Parking Assistance

Driving Assistance
Cruise Control
Lane Keep Assist

Traffic Jam Pilot

Valet Parking

Highway Pilot

Autonomous trucking

Urban Pilot / MaaS

17 June 2020
### Levels of automation and system architectures

<table>
<thead>
<tr>
<th></th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Driver 🧑‍♂️</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitoring</td>
<td>Driver 🧑‍♂️</td>
<td>Driver 🧑‍♂️</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fallback</td>
<td>Driver 🧑‍♂️</td>
<td>Driver 🧑‍♂️</td>
<td>Driver 🧑‍♂️</td>
<td>Vehicle 🚗</td>
<td>Vehicle 🚗</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(after a defined take-over time)</td>
<td>(for defined use case)</td>
<td></td>
</tr>
</tbody>
</table>

### Fail-silent system designs

- **Assumed system architectures and ASILs**
  - B
  - C
  - D

### Fail-operational system design options

- B
- B
- B
- D
- D
- D
- D
Key take-aways

A time-aware architecture supports building stable hard real-time systems and reduces complexity.

This speeds up time-to-market for new functionalities, guarantees safety, and allows software investments to be reused for highly automated driving projects.

MotionWise helps organizations move away from a slow, costly, complex and iterative integration process to a platform approach.

A service-oriented architecture with real-time guarantees provides a seamless roadmap to full automation in the future.

Eric Schmidt
Director Lead Engineering & Safety
eric.schmidt@tttech-auto.com
www.tttech-auto.com

LinkedIn