Event-chain-centric architecture design of Driver Assistance Systems

Mastering technical complexity using a cross-architectural methodology

Frieder Heckmann, Valeo Schalter und Sensoren GmbH

Ralf Münzenberger, INCHRON AG

Modern driver assistance functions are realized by the interaction of various hardware and software components. To make the ever-increasing complexity manageable, Valeo uses an architecture design focused on event chains, taking the functionality of the entire vehicle into account. This allows a dynamic architecture to be designed at an early stage of development at the logical architecture level, independent of any specific technology and validated by simulation. Integration requirements for the technical architecture can be derived from the logical architecture and verified with integration tests. This article uses the illustrative example of a driver assistance function (emergency brake assist) to demonstrate the methodology developed for this purpose.

Initial situation

Valeo develops end-customer functions for driver assistance systems that are realized through the interaction of sensors, actuators, and electronic control units (ECU). Valeo has responsibility for the overall end-customer function, i.e., the vehicle should behave as specified by the OEM (Original Equipment Manufacturer). However, Valeo only develops the functional software and, when required, the sensors, while the OEM is usually responsible for the actuators and the vehicle bus. Therefore, when considering the end-to-end behavior of the vehicle system, Valeo must also make allowance for aspects of the system for which Valeo is not responsible.

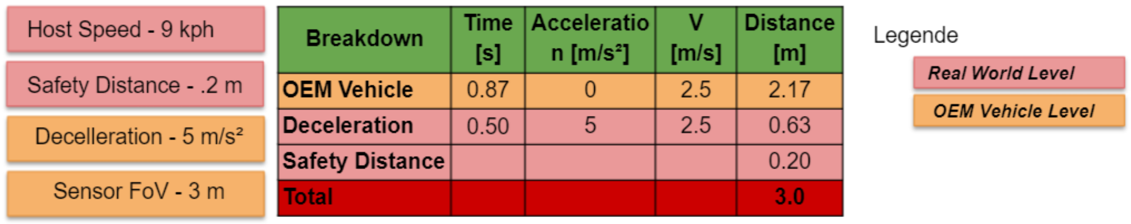
For the modeling of the end-to-end behavior, event chains have established themselves as a competent manner to model data flow at the vehicle level, abstracted from technical details, e.g., from a sensor to the actuator via one or more vehicle buses and ECUs. The event chains are also used to specify the timing requirements and budgets of the components for the event chain steps for which Valeo is responsible, as well as the timing budgets for the event chain steps for which the OEM is responsible.

Based on such event chains, a dynamic architecture is designed at the logical level. Since the timing requirements are the basis for the development of the overall function, they are verified by a real-time simulation and coordinated with the other project participants. Since event chains play a central role in the development project, we can speak of an event-chain-centric architecture design.

Derivation of the end-to-end requirements

A typical function of a low-speed maneuvering system is the emergency brake assist, as defined by NCAP. The *Car-to-Pedestrian Reverse Adult (CPRA)* use case presented below is taken from the EURO NCAP *Protocol AEB VRU systems* (EURO NCAP, 2021). It defines the conditions imposed on the vehicle and the environment but does not specify how to implement the use case. The realization depends on the vehicle, and it is the task of the architect to design the vehicle architecture in a way that the NCAP requirements can be fulfilled.

Figure 1 shows the corresponding derivation of the time budget (end-to-end latency) available to the vehicle for triggering the emergency braking. The diagram distinguishes between the real-world dimension (Pique, 2014), i.e., those properties dictated by the use case definition, and the OEM vehicle dimension, whose parameters are vehicle-specific.


Host Speed - 9 kph
Safety Distance - .2 m
Decelleration - 5 m/s²
Sensor FoV - 3 m
Legende
Real World Level
OEM Vehicle Level

Breakdown	Time [s]	Acceleration [m/s^2]	V [m/s]	Distance [m]
OEM Vehicle	0.87	0	2.5	2.17
Deceleration	0.50	5	2.5	0.63
Safety Distance				0.20
Total				3.0

Figure 1: Derivation of the real-time requirements for the vehicle from the given use case

The timing budget for the vehicle (here 0.87 s) is divided between the sensors, actuators, and ECU(s) in the next architecture design step using event chains. Typically there are empirical values available for these that can be used for an initial specification.

Modeling of an event chain

The architect specifies the emergency brake assist function as an event chain using the logical elements resulting from the functional breakdown (Hedderich, 2019) and describes the timing of the processing steps in a strictly sequential manner. Logical elements include sensors, actuators, and software (SW) blocks. These are detached from the concrete hardware (HW) platform. The event chains are defined in a model-based development tool in UML (Unified Modeling Language, 2021). The event chain can be exported automatically to a simulation tool to validate the timing requirements associated with the event chains.

Figure 2 shows an example of the modeling of the event chain of an emergency brake assist as a UML sequence diagram and the timing budgets assigned to the individual steps for sensor t_{sensor}, ECU t_{ECU}, and braking system t_{Brake}. In addition to the logical elements, it also contains two interfaces (*If* and *Bus*). These interfaces demarcate the ECU system from the sensor and the brake system, respectively. Three sensor measurements are required for reliable detection of an object. This is modeled by self-referencing the sensor with the edge label *Rep=3*.

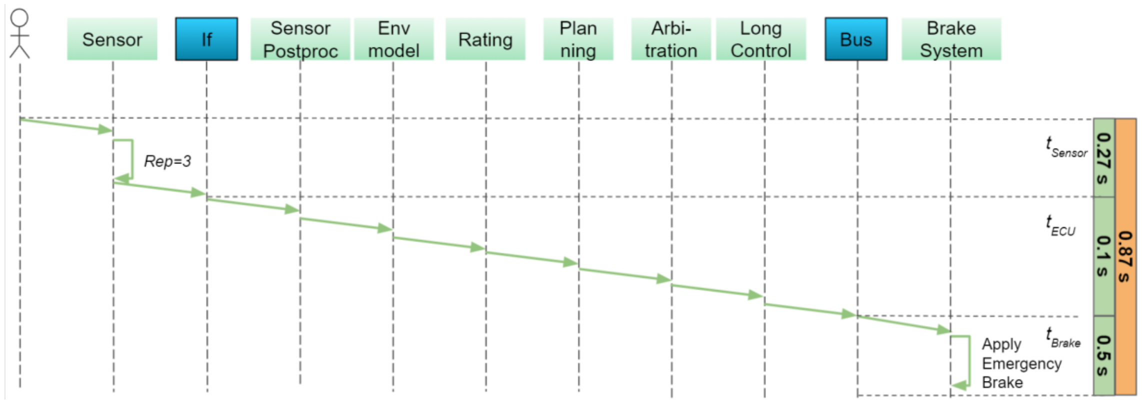


Figure 2: Example modeling of the event chain for an emergency brake assist function

Modeling elements for architecture design

The dynamic architecture includes all architecture timing constraints relevant for the specification of the temporal behavior. In the application example from Figure 2, these are the activation conditions, execution times, and delays in communication. These parameters must be defined so that the event chains meet the timing requirements while minimizing CPU load.

The logical elements are grouped into Execution Groups (EG) to simplify the design, as shown in Figure 3. An EG can be compared to a Basic Task in AUTOSAR (AUTOSAR Consortium, 2021). The logical elements correspond to runnables that the task activates sequentially. A unique feature is that the EG only processes the data that is available when activated, and the calculation results are only available when it is terminated.

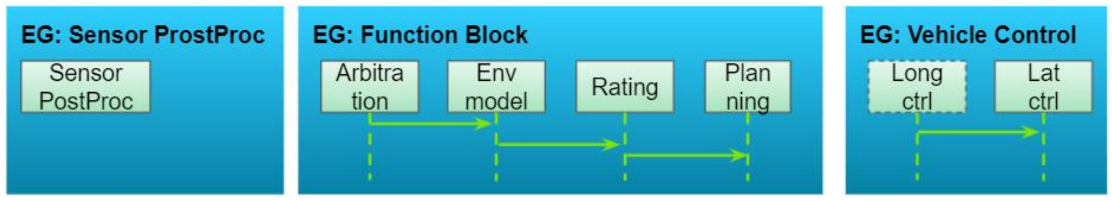


Figure 3: Structure of the Execution Groups of the Emergency Brake Assist Application Example

The concept of EGs simplifies architectural design by decreasing the number of elements to be considered and simplifies integration by allowing only pre-integrated groups to be deployed. The formation of EGs is based on the following criteria:

- Dependencies between the logical elements
- Dependency on sensor update rate
- Dependence on the vehicle bus timing

For example, *SensorPostProc* from Figure 3 forms its own EG because it depends on the update rate of the sensor and, thus, can be activated independently of the other elements.

The Environmental Model uses data from multiple sensors, including odometry data, as inputs. Since such input data typically have varying update cycles, the *Env Model* and those elements with a strong logical dependency form a single EG.

Long Control provides data for the bus. Therefore, it should be possible to activate it independently of the other EGs. The EG additionally contains a *Lat Control* element to indicate that the EG in a complex system may also contain several logical elements.

The following aspects are sufficient for modeling the dynamic behavior of an EG:

- Activation condition: Cyclic vs. event triggered.
- Max. response time of an Execution Group: Time between activation and end.
- Communication delay: Maximum latentency of communication between two EGs.

The UML profile shown in Figure 4 can be derived from these requirements, which is sufficient for describing the dynamic architecture at the logical level.

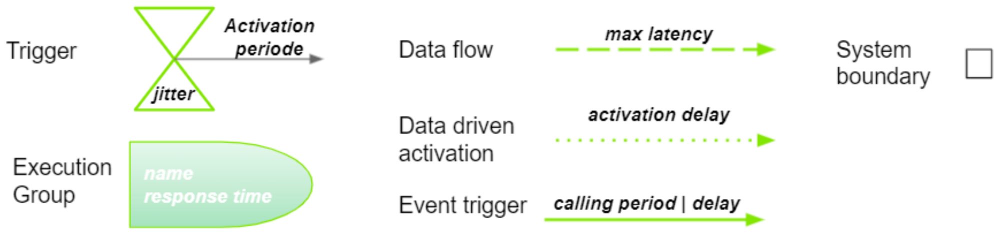


Figure 4: Elements for modeling the dynamic architecture

Design of the dynamic architecture at the logical level

Figure 5 shows a preliminary architecture conceptual design. Braking and sensing systems are modeled as EGs since they also have timing requirements. The pedestrian represents a special feature. Although not part of the system, it is required within the environment model both for simulation and requirements evaluation.

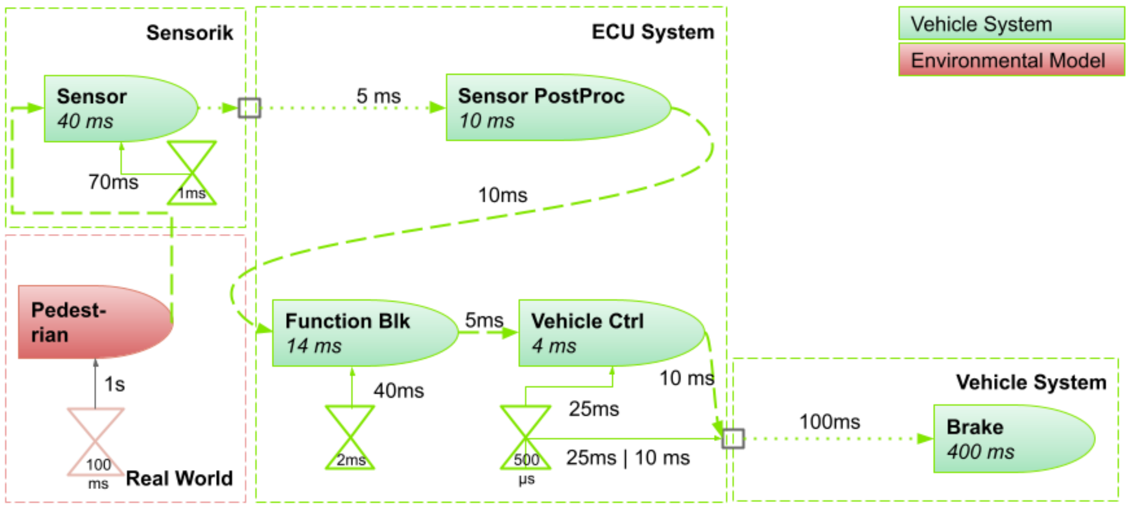


Figure 5: Graphical representation of the dynamic architecture

In this example, the sensor performs a measurement every 70 ms with a measurement jitter of 1ms. After a maximum of 40 ms, the measurement is processed and the data is provided to the ECU. The port represents the system boundary between the sensor and ECU. After a maximum of 5 ms, the ECU must trigger the EG *Sensor PostProc,* which has a maximum response time of 10 ms. The maximum acceptable delay in the data transfer to the EG *Function Block* must not exceed 10 ms. Next, the data is processed by the cyclically activated EGs *Function Block* and *Vehicle Ctrl*. The system cyclically activates the port between ECU System and Vehicle System. This simulates the bus behavior under the assumption that a bus update must take place every 25 ms. The delay on the bus and the brake system is 100 ms and 400 ms, respectively.

Verification through simulation

The event chains from Figure 2 and the architecture from Figure 5 can be transferred to a simulation model. This makes it possible to simulate the dynamic system behavior and check it against the real-time requirements. Valeo uses chronSIM (INCHRON AG, 2021) for this purpose, which performs the validation automatically.

Figure 6 shows the event chain (top), as defined in Figure 2, together with the activation of the individual EGs encapsulating the elements used in the event chain, including the pedestrian relevant for the activation. Simulations with the real-time simulator chronSIM determine the latency for each event chain pass and compare it against the requirement from the analysis shown in Figure 2.

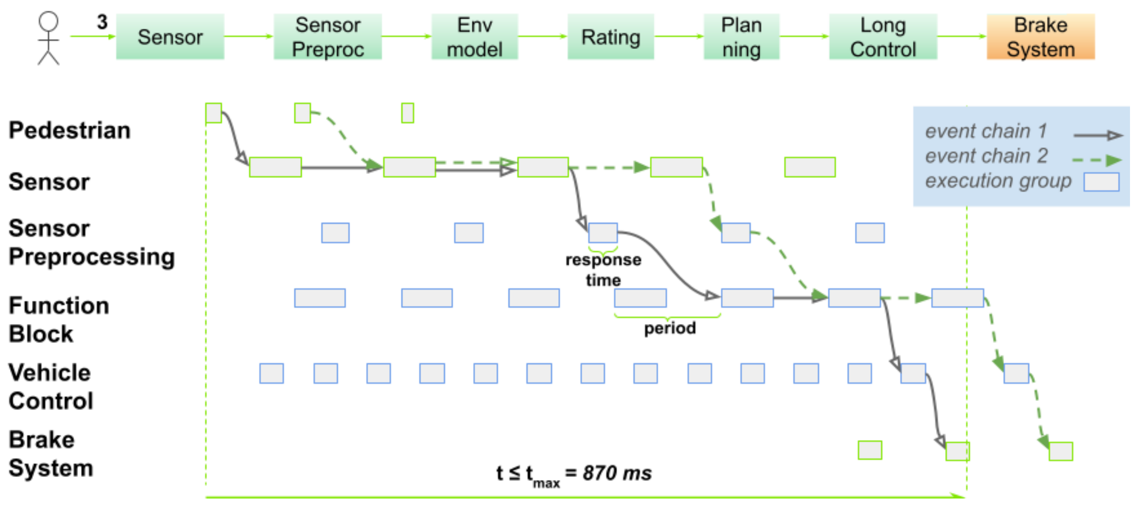


Figure 6: Simulation of the dynamic architecture including event chains

Figure 7 plots the latencies determined from the simulation as a histogram and shows that the majority of the simulated event chains violate the timing requirements. The histogram shows the distribution of the event chain latency. It exhibits a wide spread because the jitter of the individual timer elements leads to changing latencies between the EGs that influence the overall end-to-end latency. Since the architecture design exceeds the maximum latency of 870ms it must be optimized.

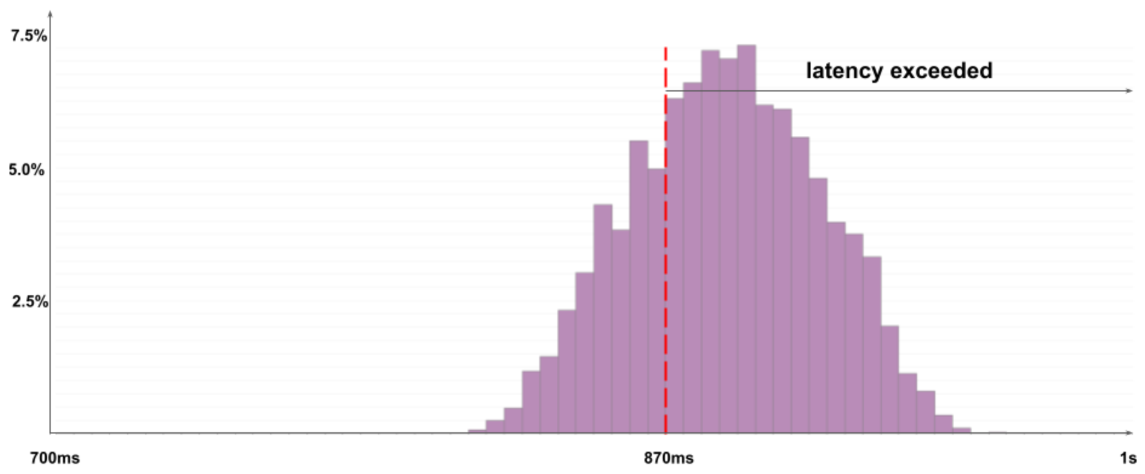


Figure 7: Histogram of event chain latencies

Optimization step

The evaluation of the simulation in Figure 6 shows that the event chain is routed through two successive instances of the EG *Function Block* before the result is available for the EG *Vehicle Control*. This is because, in the event chain definition in Figure 2, the logical element *Arbitration* follows *Rating*. However, in the logical architecture, the activation of *Arbitration* in the EG *FunctionBlk* occurs before *Rating*. Moving the *Arbitration* component to the end of the EG execution sequence fixes this problem.

Figure 8 shows an alternative architecture design that contains some further optimizations. *Vehicle Ctrl* triggers *Function Block* every second activation, reducing the latency between the two EGs compared to the first design. The response time of *Vehicle Ctrl* and *Function Blk,* as well as the communication delay between these EGs and the *Bus*, was reduced in order to fulfill the timing requirements given by the event chain.

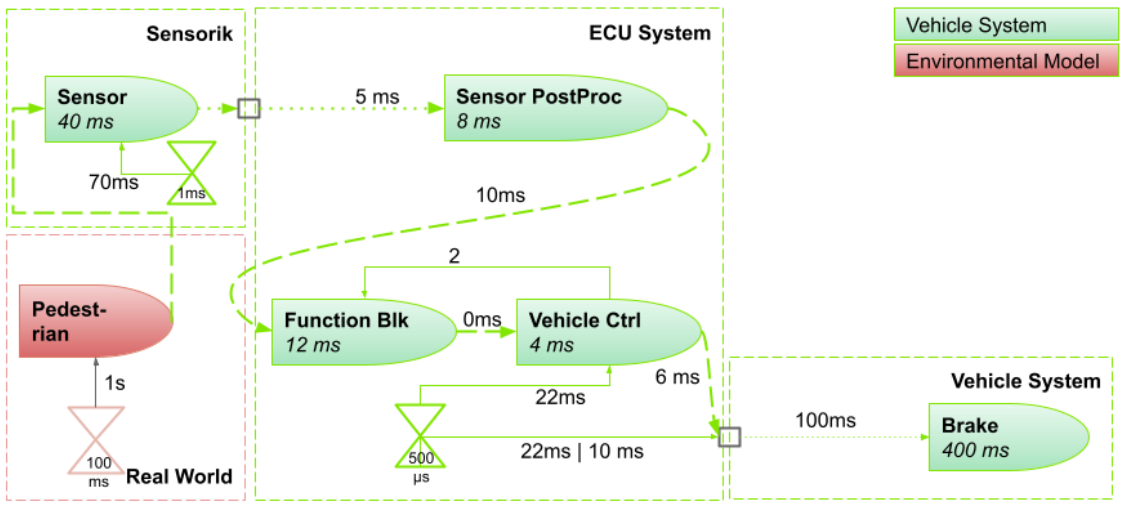


Figure 8: Optimized architectural design

Figure 9 shows the distribution of the end-to-end latencies of the event chains as a histogram in chronSIM after the achitectural optimization. The maximum latency of 870 ms is no longer exceeded, and the spread of the distribution is narrower compared to the results from Figure 7.

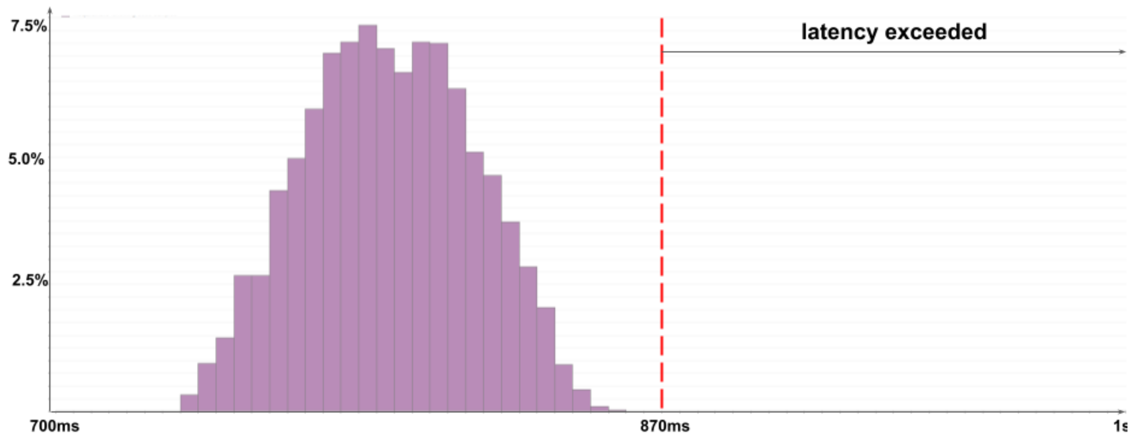


Figure 9: Histogram of the event chain latencies after optimization

Deriving the timing requirements for integration testing

Timing errors are challenging to find in the vehicle or HIL test since they only show up indirectly through faulty functional behavior, and often only occur sporadically. Therefore, it is crucial to test the timing integration requirements continuously and on a broad basis. The integration requirements can be derived directly from the logical architecture (Figure 8) and include:

- The response time of the EGs,
- the activation requirements of the EGs,
- and the communication delay between the EGs.

If the technical architecture meets these requirements, it also executes the event chains within the defined timing budget. This could be evaluated using a timing model of the technical architecture with chronSIM.

In order to execute such tests on the target, the system requires around 5 kBit/s bandwidth to record the timing data needed for an evaluation of the integration requirements, as shown in Figure 8. This data can be output via the vehicle bus, and recording is possible without additional HW effort (within a test fleet).

Tools like chronVIEW (INCHRON AG, 2021) can analyze the traces automatically and check them against the requirements of the logical architecture. Thus, large data sets can be evaluated and can detect even rarely occurring errors in timing behavior.

Summary

The design of a robust architecture for a driver assistance system is only possible by considering the end-to-end behavior of the entire function. The event-chain-centered architecture design presented in this article is a method proven in practice to specify the real-time requirements and use them to develop a dynamic architecture at the logical level. Automated validation of the architecture design helps to make the system manageable, as this is the only way to automatically test many event chains such as those found in complex driver assistance systems.

The requirements for the technical architecture and criteria for the system integration test can be derived from the logical architecture. This ensures consistency between design and testing in a project since the requirements originate from a single source.

References

AUTOSAR Consortium. (2021). *AUTOSAR*. Von AUTOSAR: https://www.autosar.org/ abgerufen

EURO NCAP. (2021). *TEST PROTOCOL – AEB VRU systems.*

Hedderich, M. a. (2019). SEMAS – System Engineering Methodology for Automated Systems | The world described in layers. *MBMV 2019; 22nd Workshop - Methods and Description Languages for Modelling and Verification of Circuits and Systems*.

INCHRON AG. (2021). *INCHRON*. Von www.inchron.com/chronVIEW abgerufen

Pique, J.-D. (2014). SysCARS – SysML for embedded automotive systems. *ERTS*.