Dr. Eric Armengaud
VIF - Area E
Group leader embedded systems

January 10th, 2011

Model-based development and test of distributed automotive embedded systems
Electronics in car

Software Engineering for automotive embedded systems

The time-triggered architecture & FlexRay

Research activities @ Virtual Vehicle Competence Center
**VIRTUAL VEHICLE in a nutshell:**

<table>
<thead>
<tr>
<th>Founded:</th>
<th>July 2002</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Staff:</td>
<td>150</td>
</tr>
<tr>
<td>Turnover:</td>
<td>EUR 12 Mio.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shareholder:</th>
</tr>
</thead>
<tbody>
<tr>
<td>TU Graz</td>
</tr>
<tr>
<td>AVL</td>
</tr>
<tr>
<td>MAGNA STEYR</td>
</tr>
<tr>
<td>SIEMENS</td>
</tr>
<tr>
<td>JOANNEUM RESEARCH</td>
</tr>
</tbody>
</table>

- **Managing Director:** Dr. Jost Bernasch
- **Scientific Director:** Prof. Hermann Steffan
  (Vehicle Safety / Frank Stronach Institute, TU Graz)
Independent Research Platform
(not tied to specific bodies or corporations)

Applied Research and Scientific Services

Driven by the demand of leading companies
(> 50 industry partners)

Comprehensive international Research Network
(> 35 scientific partners and university institutes)

Extensive financial funding programs available
(no overhead as in customary funded projects)
“If the automotive industry had advanced as rapidly as the semiconductor industry we’d all be **driving a Rolls Royce**, it would do **half a million miles to the gallon** and it would be **cheaper to throw away than to park**.”

And as a friend pointed out, Moore said, "it would only be a **half-inch long and a quarter-inch high**."
Vehicles a decade ago
- A few embedded systems per vehicle

Vehicles nowadays
- Up to a few hundreds of computing devices per vehicle
- Multiple networks per vehicle

Advantage
- Safety-critical embedded systems have been key innovation drivers
- E.g. by-wire systems

Disadvantage
- Enormous complexity is challenging industry (automotive, aerospace, rail, automation)
- Increasing costs
- Affected product quality ➔ safety-critical
R&D spending in automotive industry:

- In 2005 € 68 billion in research & development
  • 4.2 % of sales or € 783 per vehicle
- Additionally € 1.500 of cost reduction per vehicle forecasted (11 % of costs)
- Through 2015, R&D will rise to € 800 billion
- E&E will remain the most important enabler for automotive innovations

[H. Gall, austriamicrosystems]
80% of the innovation in cars comes from electronics – Software plays a key role

Evolution of the complexity
- Programming language: assembler in the 1970’s, C in the 90’s, Matlab/Simulink (ASCET) nowadays
- 100 millions lines of code
- up to 80 ECUs
- 2500 signals
- 65 millions cars and light commercial vehicles produced each year
- Large development teams regrouping different domains and different institutions

Requirements on automotive electronics
- High reliability
- Functional safety
- Real-time behavior
- Minimized resource consumption
- Robust design
But, different kinds of models for different skills (dysfunctional models for safety, performance models for timing constraints, …)

[CESAR Project, O. Laurent, AIRBUS]
Needs for improved development processes

Methods for requirement engineering
- First description of the system; contract between OEM and supplier
- Requirements needs to be precise, unambiguous and complete
- Formalization of multi viewpoint, multi criteria and multi level requirements

Methods for component-based design
- Global understanding of the system for efficient analysis
- Provide traceability during system design and validation
- Design space exploration comprising multi-view, multi-criteria and multi level architecture trade-offs

Safety methods and processes
- Ensure the quality of a product via the execution of safety related activities and the definition of a standardized development process
- Provide traceability of the development process
- Formalization of the dev. process for analysis, reporting (certification) and automation (service orchestration)
Software Engineering

“Software engineering (SE) is a profession dedicated to designing, implementing, and modifying software so that it is of higher quality, more affordable, maintainable, and faster to build. It is a systematic approach to the analysis, design, assessment, implementation, test, maintenance and re-engineering of a software by applying engineering to the software”

[Wikipedia]
Requirements engineering:
- deals with understanding, documenting, communicating and implementing customer needs
- is required to reach a common understanding between the stakeholders
- is required during the entire development cycle (design, implementation, validation)

Related activities
- **Requirements elicitation**: find out the services the system should provide and the operational constraints
- **Requirements analysis and negotiation**: solve the conflicts between the requirements in order to reach a common understanding between the stakeholders
- **Requirements documentation and validation**: write down and check the requirements against correctness, completeness, consistency, verifiability, unambiguity, traceability…
- **Requirements management**: managing requirements changes (keep the requirement set consistent)
Requirements Engineering

Requirement Specification Language

- minimize amount of time to write requirements
- make requirements understandable and unambiguous
- minimize amount of time to validate requirements
- differences in:
  - formality: formal/semi-formal/informal
  - illustration: textual/graphical/tabular

Requirement Meta-Model

- capturing, managing and organizing requirements into a formalized structure
- providing the meta-model for each RSL
- providing the interoperability model for tools

[MEPAS Project, N. Marko]
Free text: no constraint  
→ no training required  
e.g.: the system shall count time between eyelid movement and warn driver if the time is less than 2 sec

Guided natural language: limited vocabulary from a dictionary  
→ reduce ambiguity  
e.g.: **driver**: person who drives the car // **warn**: inform the driver about an event

Structured textual: template for requirement description  
→ further reduce ambiguity, support transition to formal notations  
e.g. IF <trigger> THEN <subject> SHALL DO <action list> WITHIN <time bound>

Semi-formal model-based: formal and precise syntax while their semantics are imprecise and allow different interpretation  
→ support the analysis of the requirements  
e.g.: UML modeling

Formal model-based: method for definite, orderly and methodical requirement definition  
→ most precise requirement definition  
e.g. Petri nets, timed automata
Traceability: „Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction, …“

post-requirements traceability links
- satisfies
- verify
- realize

pre-requirements traceability links
- explicit traceability links
  - owns
  - hasRationale
  - hasSource
- possible operations performed on requirements
  - refine
  - decompose
  - copy
  - depend
Motivation

- Describe in a formalized way the different artefacts of a system
  ➔ improved specification of the system
- Explicitly link the different artefacts together
  ➔ improved analysis and optimization capabilities of the system
- Provide a computer-based framework
  ➔ Support engineers during development activities and improve tool interaction

Some (non-functional) modeling languages for automotive domain

- EAST-ADL: architecture description language tailored for the automotive domain (www.atesst.org)
- AUTOSAR: AUTomotiv Open System Architecture (www.autosar.org)
- FIBEX: Field Bus Exchange Format (www.asam.net)
- TIMMO: Timing Model (www.timmo.org)

A modeling language supports your development work but will NEVER take away the intellectual work of creating and understanding your system
The Multi-views approach: the double challenge

The first challenge: Identify the relevant views

possible views:
• Operational: focus on the system missions
• Functional: focus on the functional aspects of the system
• Logical: define system architecture, define abstract components, allocation of functions on them, behaviour of components and interfaces between components
• Physical: define concrete hardware and software components, allocation of functions on hardware and software components
• Safety: define the dysfunctional aspects of the system
• Product line: define the variability points
• Performance: define the system performance
• Interface: define the interfaces of the system components

The second challenge: Propose the appropriate foundations to share common data between the different views

The multi-views prism

[CESAR Project, O. Laurent, AIRBUS]
EAST-ADL is an architecture description language with improved means for capturing the requirements, characteristics and configurations of cooperative systems and the related analysis and V&V.
AUTOSAR (AUtomotive Open System ARchitecture) is an open and standardized automotive software architecture, jointly developed by automobile manufacturers, suppliers and tool developers.
FIBEX is an XML-based standardised format used for representing the networks used in the automobile. It has extensibility required for the various network protocols used.

**Topology:** ECUs, comm. channels, HW types

**Communication matrix:** mapping between data models (signal-PDU-frames), timing information

**Application:** signals, variable
TIMMO develops a common, standardized infrastructure for the handling of timing information during the design of embedded real-time systems in the automotive industry. This shortens the development cycle and increases its predictability.
Seamless modeling - vision

Requirements

system architecture specification and management (EAST-ADL)

Safety analysis (HiP-HOPS)

behavioral modeling (Matlab / Simulink)

Further static analysis

architecture modeling (AUTOSAR)

IDE
The ECUs are deployed for safety-relevant operations (e.g., car movement, power distribution, vehicle stability), where a failure can harm people, environment or property and has therefore to be avoided.

IEC 61508: "Functional safety of electrical/electronic/programmable electronic safety-related systems"

- Basic functional safety standard applicable to all kinds of industry
- Published 1998, since 2001 as European norm
- Covers the entire development cycle (16 phases covering analysis, realization, operation
- Central concepts *risk* and *safety function*
- Philosophy
  - zero risk can never be reached
  - safety must be considered from the beginning
  - non-tolerable risks must be reduced (ALARP - as low as reasonably practicable)
ISO 26262: “Road vehicles – Functional safety” ([www.iso.org](http://www.iso.org))
- Based on IEC 65801
- Defines safety process for the development of road vehicles
- Draft International Standard – will be released in 2011

Safety
- Freedom from unacceptable risk
- Risk: combination between probability and severity of a failure

Safety related project activities
- **Risk**: Hazard and risk analysis
  e.g. “what could happen if”
- **Safety**: safety concept
  e.g. “what is the safe state”
- **Safety functions**: safety requirements
  e.g. “How to provide the safe state”
- **SIL decomposition**: Implementation and processes
  e.g. “what SIL (Safety Integrity Level) applies for individual units”
Standardized development processes including safety-related activities

Accompanying processes

ISO 26262

<table>
<thead>
<tr>
<th>Vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Management of Functional Safety</td>
</tr>
<tr>
<td>2-5 Overall Safety Manangement</td>
</tr>
<tr>
<td>2-6 Safety Management during Item Development</td>
</tr>
<tr>
<td>2-7 Safety Management after Release for Production</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3. Concept Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-5 Item Definition</td>
</tr>
<tr>
<td>3-6 Initiation of the Safety Lifecycle</td>
</tr>
<tr>
<td>3-7 Hazard Analysis and Risk Assessment</td>
</tr>
<tr>
<td>3-8 Functional Safety Concept</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. Product Development: System Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-5 Initiation of Product Development at System Level</td>
</tr>
<tr>
<td>4-6 Specification of the Technical Safety Requirements</td>
</tr>
<tr>
<td>4-7 System Design</td>
</tr>
<tr>
<td>4-8 Item Integration and Testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. Product Development: Hardware Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-5 Initiation of Product Development at the Hardware Level</td>
</tr>
<tr>
<td>5-6 Specification of the Hardware Requirements</td>
</tr>
<tr>
<td>5-7 Hardware Design</td>
</tr>
<tr>
<td>5-8 Hardware Architectural Metrics</td>
</tr>
<tr>
<td>5-9 Evaluation of Violation of the Safety Goal due to random HW Failures</td>
</tr>
<tr>
<td>5-10 Hardware Integration and Testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6. Product Development: Software Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>6-5 Initiation of Product Development at the Software Level</td>
</tr>
<tr>
<td>6-6 Specification of the Software Requirements</td>
</tr>
<tr>
<td>6-7 Software Architectural Design</td>
</tr>
<tr>
<td>6-8 Software Unit Design and Implementation</td>
</tr>
<tr>
<td>6-9 Software Unit Testing</td>
</tr>
<tr>
<td>6-10 Software Integration and Testing</td>
</tr>
<tr>
<td>6-11 Verification of Software Safety Requirements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. Production and Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-5 Production</td>
</tr>
<tr>
<td>7-5 Operation, Service, (Maintenance and Repair), and Decommissioning</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>8. Supporting Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-5 Interfaces with Distributed Developments</td>
</tr>
<tr>
<td>8-6 Specification and Management of Safety Requirements</td>
</tr>
<tr>
<td>8-7 Configuration Management</td>
</tr>
<tr>
<td>8-8 Change Management</td>
</tr>
<tr>
<td>8-9 Verification</td>
</tr>
<tr>
<td>8-10 Documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. ASIL Oriented and Safety Oriented Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>9-5 Requirements Decomposition with Respect to ASIL Tailoring</td>
</tr>
<tr>
<td>9-6 Criteria for Coexistence of Elements</td>
</tr>
<tr>
<td>9-7 Analysis of Dependent Failures</td>
</tr>
<tr>
<td>9-8 Safety Analysis</td>
</tr>
</tbody>
</table>

| 10. Guideline to ISO 26262 (informative) |

ISO DIS 26262
Challenges

Safety concept
- Is required to analyze systematically the risks of the controlled system in its environment
- Provides additional requirements to the system
- Has a direct influence on the system architecture and functionality (safety functions)

Traceability of the development process
- It must be ensured that the standardized development process has been followed (audit from external companies)
- The tool chain must be reasonably reliable (classification and qualification activities)

Needs for Software Engineering
- Development activities are part of the ISO 26262 (“What” - which kind of test, review, analysis)
- However specific development methods are not part of the ISO 26262 (“How”)
- Systematic approach for designing, implementing, and modifying the software is required to improve system quality while minimizing the costs
Time-triggered architectures for complex control applications

“… in the **event-triggered** approach, all communication and processing activities are initiated whenever a significant change of state, i.e., an event (e.g., interrupt), is noted. In the **time-triggered** approach, all communication and processing activities are initiated at predetermined points in time.”

Example: Volkswagen Golf

1976: Golf I
0 ECU’s

1983: Golf II
5 ECU’s

1991: Golf III
11 ECU’s

1998: Golf IV
18 ECU’s

2004: Golf V
Up to 35 ECU’s

2006: Golf VI
Up to 48 ECU’s

© VIRTUAL VEHICLE
Snapshot 2004: the VW Phaeton

- 2110 cables
- 3860 meters cable
- Weight: 64kg
- 70 ECUs

Advantage

- Safety-critical embedded systems have been key innovation drivers
- E.g. by-wire systems

Disadvantage

- Enormous complexity is challenging industry (automotive, aerospace, rail, automation)
- Increasing costs
- Affected product quality ➔ safety-critical

Source: Volkswagen Beetle, 1960
Source: Technology review, July 2004
Automotive communication networks

- **SAE Class A**
  - Comfort and simple control
  - ~10kB/s
  - Low cost technology

- **SAE Class B**
  - 10 - 125kB/s
  - Data exchange between ECU

- **SAE Class C**
  - 125kB/s – 1 MB/s
  - High speed control loops
  - Powertrain, chassis
  - (Hard real-time)

- **SAE Class D**
  - >1MB/s
  - Multimedia
  - Soft real-time

- **CAN**

- **J1850**

- **SAE Class D**

- **MOST**
  - Ethernet
  - IEEE1394

- **FlexRay™**

- **USB**

- **Relative price per node**

- **Data rate (bit/s)**
  - 100M
  - 10M
  - 1M
  - 20K
Needs for new architectures

**Automotive electronics organized as complex distributed systems**
- Local connection between sensors, processors and actuators
- Information dissemination within the car
- Point to point connection inefficient (reliability, weight)

**System complexity difficult to manage**
- Number of ECU, intensity of the communication
- Different technologies
- Complexity of the application

**The system can not be assumed fault-free**
- High temperature range and thermal gradients
- High humidity, splashes from oil, petrol, chemicals…
- Conducted emissions (electric motors) and radiated emissions (power lines, radio or TV transmitters)
Event-triggered architecture

- System activity triggered by an event
- Priority based communication (CAN)

- Communication jitter
- Constructive integration
- Redundancy
- Architecture flexibility
- Bandwidth use (sporadic events)
Time-triggered architecture

- Action derived from progression of time
- Static, periodic, a-priori known schedule
- Global notion of time

Communication jitter

Constructive integration

Redundancy, Agreement

Architecture flexibility

Bandwidth use (sporadic events)
ET versus TT Transmission paradigm

Event-based communication
- A communication is triggered for each new event – i.e. major state change (e.g. temperature increase of +5 degree)
- Each event (communication) has to be detected and processed in the same time order it arrived
- Optimal use of the bandwidth
- Not robust – lost of message might lead to system inconsistencies

Status-based communication
- Periodic communication for updating system state (e.g. temperature is currently 55 degree)
- Events (communication elements) might be missed or processed in different time order than reception time
- Worse-case use of the bandwidth
- Robustness: lost of message only induce additional processing delays – no system inconsistencies
FlexRay Overview
Periodical communication scheme

- Static segment for time-triggered communication
- Dynamic segment for event-triggered communication
- Symbol window for medium test
- Network idle time for resynchronization
Medium access

Static segment: static schedule for time-triggered communication

Dynamic segment: prioritized access for event-triggered communication
FlexRay controller overview

Controller Host Interface:
- Data exchange with host
- Data exchange with communication controller

Clock synchronization:
- Measurement of the clock (state and rate) differences
- Computation of a fault-tolerant correction term
- Provides both rate and offset correction

Coding / Decoding:
- Signal processing
- Adding / removing bit sequences
- Syntax check

Media access control:
- Generate communication schedule
- Provide time information (segment, slot…)

Macrotick generation:
- Macrotick: commonly agreed time base within the network
- Required to generate the schedule

Protocol operation control:
- Configuration: provides mechanisms for configuration
- Control: control the protocol state (stopped, normal, error…)

Frame and symbol processing:
- Packing / unpacking contents into frames
- Error check (content)

Coding & decoding ch. A
- Frame and symbol processing
- Media access control

Controller host interface
- Protocol operation control
- Clock synchronization

[FlexRay Protocol Specification, V.2.1A, Fig 2-2]
Aim: provide a global time base within the network to correct the quartz drift and avoid collision on the bus.

Requirement: fault tolerant algorithm

- No single point of error
- Single faults are discarded
Clock synchronization - overview

Goal
- Synchronize the macroticks between the nodes
- Keep the system precision (maximal time difference between any two nodes) bounded

Offset correction
- Goal: minimize the clock state difference at cycle start
- Correct the number of microticks per cycle
- Discrete correction (once per cycle)

Rate correction
- Goal: minimize the clock state difference within the cycle
- Modify the number of microticks per macroticks
- Continuous correction
Motivation
- Wake-up the network and provide initial synchronization
- Fault tolerant (network operation relies on start-up)
- Fast operation (fault recovery)

Three phases
- Wakeup: to wake-up the network (active stars, nodes) if it is still asleep
- Startup: to begin communication (initialize schedule) when the nodes are awake
- Reintegration: to integrate single nodes within a running cluster
Wakeup - illustration

Node’s state machine

Node A
- coldstart node
  - reset
  - config
  - ready
  - Wup listen
  - Wup send
  - ready
  - integration listen
  - coldstart listen

Node B
- coldstart node
  - power off / reset
  - config
  - ready
  - Wup listen
  - Wup send
  - ready
  - coldstart listen

Node C
- non coldstart node
  - power off / reset
  - config
  - ready
  - integration listen

Channel
- A
  - wakeup pattern

- B
  - wakeup pattern

[FlexRay Protocol Specification, V.2.1A, Fig 7-6]
Node’s state machine

- **Node A**:
  - Leading coldstart
  - State transitions:
    - Ready
    - Coldstart
    - Coldstart collision resolution
    - Consistency check
    - Normal active

- **Node B**:
  - Following coldstart
  - State transitions:
    - Ready
    - Coldstart
    - Initialize
    - Integration coldstart check
    - Coldstart join
    - Normal active

- **Node C**:
  - Non coldstart node
  - State transitions:
    - Ready
    - Integration
    - Initialize
    - Integration consistency check
    - Normal active

- **Cycle schedule**:
  - No schedule
  - Cycle 0 to 8

- **Channel**:
  - CAS S S S S SS SS SS SS SS
  - A A A A AB AB AB AB AB AB

[FlexRay Protocol Specification, V.2.1A, Fig 7-10]
Integration issues
Design of the communication architecture - Fibex

**Topology**: ECUs, comm. channels, HW types

**Communication matrix**: mapping between data models (signal-PDU-frames), timing information

**Application**: signals, variable

[FIBEX - Field Bus Exchange Format, Version 3.0 ASAM AE, 2008, Fig 10-1]
Integration issues within the software architecture

Control flow – integration within the SW architecture

- **Event-triggered** (action triggered with rxd / txd interrupt) flexibility but control flow difficult to handle (interrupts)
- **Time-triggered** (comm. task synchronous to bus schedule) a-priori known behavior (time domain) but complex dependencies between operating system and communication system

Data flow – transmission scheme

- **Buffer**: frame filtering (ID, cycle) performed in hardware, Time-triggered comm.: **latest data stored** (old version discarded)
- **FIFO**: frames stored sequentially, further processing in software Event-triggered comm.: **all frames are available**

System configuration – amount of data

- **Protocol configuration**: FlexRay schedule / syntax (>70 param.)
- **Data access and interpretation**: buffer configuration, mapping between frame and signals (>50 parameters)
## Interaction between Operating and Communication system

### Operating system

<table>
<thead>
<tr>
<th>Communication system</th>
<th>Event-triggered (interrupts driven)</th>
<th>Time-triggered (schedule)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event-triggered</strong> (e.g. CAN)</td>
<td>➔ <em>Priority based communication</em> + Flexibility, average response time - Complex timing analysis, - No constructive integration</td>
<td>➔ Static communication scheme supported by the application + Easy timing analysis - Application overhead (e.g. for synchronization)</td>
</tr>
<tr>
<td><strong>Time-triggered</strong> (e.g. FlexRay)</td>
<td>➔ <em>static comm. scheme with interrupt based data interface</em> + constructive integration (comm. point of view) - Complex end-to-end timing analysis - No constructive integration (node’s point of view)</td>
<td>➔ <em>Asynchronous systems</em> + Easy timing analysis - Non optimal end-to-end delays (synchronization) - Frames might be missed</td>
</tr>
</tbody>
</table>

- *Synchronous systems* + Easy timing analysis + deterministic and optimal end-to-end delays - Flexibility
Cars are forming complex distributed systems, evolving in harsh environments; in parallel their reliability requirements increase

Automotive embedded systems from two perspectives

- Software engineering (requirement engineering, model-based development and functional safety)
- System architecture (event- vs. time-triggered)

The question is not anymore “how can I develop a given function” but “how can I make my system more dependable for lower costs”

- Meta-information for the description of the product are important
- Traceability between the system views required
- Traceability of the development process required
- (model-based) tool-chain as central elements to achieve these goals

Do not forget Verification and Validation activities!
Thank you for your attention!

www.v2c2.at