Towards model-based integration of component-based automotive software systems

Johannes Schlatow, Mischa Möstl and Rolf Ernst, Marcus Nolte, Inga Jatzkowski and Markus Maurer

Oct. 30, 2017
Motivation

Driving factors

- ADAS / automated driving
- complex high-performance architectures (Audi zFAS, NVIDIA Drive PX)
- complex software systems
  - component-based/service-oriented design
- in-field deployment of changes (features, updates)

Challenges

- non-functional requirements (e.g. safety) cannot be verified on interface basis alone (non-composability)
- in-depth understanding of different system layers
- automation/tooling → model-based integration

[Source: NVIDIA]
Background: Controlling Concurrent Change (CCC)

- research project at TU Braunschweig funded by DFG (Germany Research Foundation), 6yrs
- application scenarios: ADAS and space robots

Idea:

From
- Design & Implementation (platform-independent)

To
- Design & Impl.

Integration & Testing (lab-based)
Integration & Verification (platform-dependent)

Operation (in-field)

http://ccc-project.org
Update problem:

“Here, I have a new functionality for you. Can you please install it on your platform?”
CCC architectural approach

Multi-Change Controller (MCC)

Software Component

Application Shaper/Monitor

Run-Time Environment (including OS)

Platform Shaper/Monitor

Hardware Component

Network

model domain
at down time

execution domain
at run time
CCC architectural approach

This talk.

model domain at down time

execution domain at run time
Model domain and Multi-Change Controller

**Given:**
- component repository (incl. models, requirements, dependencies, etc.)
- update query

**Wanted:**
- a corresponding system configuration
- s.t. requirements and constraints (safety, real-time, etc.)

**Modelling challenges:**
- What components can I use (are used) to implement this functionality?
- In what functionality is this component involved?
- What is the sensing-to-actuation delay for XY?
- Are functions A and B sufficiently independent?
- ... 

→ multi-layered and graph-based modelling approach
Cross-Layer Integration

- layer = directed graph
- integration = graph transformations

**Basic transformations**

- Arc splitting (Def. in paper)
  
  ![Diagram of arc splitting](image)

- Pattern-based transformation (Def. in paper)
  
  ![Diagram of pattern-based transformation](image)
Layer overview

**Modelled entities + relations**

- **functional architecture**
  - platform independent

- **communication architecture**
  - implementation independent

- **component architecture**
  - implementation dependent

- **component instantiation**

- **functional blocks**
  + dependencies

- **functional blocks**
  + communication

- **components**
  + service dependencies

- **component instantiations**
  + connections
Example: inertial navigation system

lateral guidance + longitudinal guidance

target platform

ECU1

ECU2

network

IMU

ECU1

CAMERA

RADAR

IMU: inertial measurement unit
Function architecture $\rightarrow$ communication architecture

Query (= functional architecture subgraph):

**functional blocks**

- IMU
- Lat.
- Steering

$G_{func}$

**arc splitting:**

- mapping + reachability

**functional blocks**

- IMU
- Lat.
- Steering

$G_{comm}$

Lat.: Lateral Guidance
Component architecture $\rightarrow$ component architecture

- **pattern-based transformation**
- **arc splitting**: compatibility + cardinality

- **functional blocks + communication**
  - IMU
  - Lat.
  - Steering

- **components + service dependencies**
  - ECU1
  - NIC1
  - LET rx
  - IMU drv
  - Lat.
  - Steering

  - ECU2
  - NIC2
  - muxer
  - LET tx
  - Traj.Calc.

- LET: logical execution time
- NIC: network interface controller

** drv:** driver component
Component architecture $\rightarrow$ component architecture

**functional blocks + communication**
- IMU
- Lat.
- Steering

**components + service dependencies**
- ECU1
- ECU2
- NIC1
- LET rx
- NIC2
- muxer
- LET tx
- Traj.Calc.

- IMU drv
- Lat.
- Steering

Diagram elements:
- $G_{comm}$
- pattern-based transformation
- arc splitting: compatibility + cardinality

**Definitions**
- drv: driver component
- LET: logical execution time
- NIC: network interface controller

Component instantiation

Component architecture
lateral guidance

Component architecture
longitudinal guidance
(similar process)

ACC: adaptive cruise control
Long.: Longitudinal Guidance
Summary

- **in-field** automated integration of software (updates) for vehicles
- **multi-layer** modelling for component-based systems
- integration = graph transformation + **synthesis**
- enables **tracking** of relations and dependencies across layers
  → essential for verification
- still requires methods and mechanisms for **automating design decisions**
- subject to **non-functional requirements** (e.g. latency)

Questions?