View-Centric Modeling of Automotive Logical Architectures

Hans Grönniger, Holger Krahn, Bernhard Rumpe
Software Systems Engineering
Technische Universität Braunschweig

ISIS, Vanderbilt University
Nashville
Motivation

- **Innovations** in automotive system mostly achieved by implementing **new software functions**
- Experts estimate 1GB of onboard software in premium class vehicles by 2010
- Large number of functions from different vehicle domains with **complex interactions**
- **Methodologies** needed to break down complexity into manageable tasks
- **Processes and tools** needed for seamless development
- Developing automotive embedded system is complex
  - Many challenges, we concentrate on modeling **functional properties** of the system
Development Process

- Several divisions / teams are responsible for various features
  - Feature = functionality perceptible by customers and/or engineer (e.g. braking system)
  - aka service

- Requirements for features captured mainly textually (using e.g. DOORS)

- Specification of software architecture, description of hardware architecture (ECUs and busses), mapping
Logical architecture

- Important step in between: **logical architecture**
  - **Abstract description of functionality**
    - supports design decisions
    - specifying software components involves aggregating or splitting logical functions into software functions with exact interfaces
      - hard without explicit logical architecture
  - Less coordination effort
  - Early functional integration

- **Function Net Model**
  - Network of functions
  - Function communicate by exchanging signals
Why function nets?

- More **abstract** than software architecture descriptions (no technical details)
  - Exact data types, idealized computation, communication
- Allow for establishing a **shared understanding** of the system
  - Stakeholders with different professional backgrounds
    - Function developer
    - Safety engineers
    - Maintenance staff
    - Suppliers, ...
- Allow deriving
  - Requirements on infrastructure
  - Resources: memory, CPU time
- Commonly accepted notation necessary
  - in line with existing standards
How to use function nets in line with existing standards?

- Various options
  - UML, UML-RT, SysML, AUTOSAR-SWA, other ADLs

- SysML (Internal) Block Diagrams suitable for function net modeling
  - SysML uses *systems engineering terminology* (not OO)
  - no strict two-layer hierarchy required
  - cross-hierarchy communication links missing

- SysML more or less just notation
  - still, *customization of notation* needed to adapt to specific needs
  - develop principles of using SysML block diagrams in a suitable way
Example block diagram

Hierarchical modeling of blocks (here 3 layers)

Cross-hierarchy communication (breaks encapsulation but avoids port delegation)

No hierarchical name space

Only directed connectors

No Multiplicities (exactly one)
**Definition and Re-Use of a Block:**
- Only existing instance of that type by default
- Not point of re-use by default (as in OO)
- For re-use: draw separate block diagram and “instantiate” block

**Definition and Re-Use of a Signal:**
- Only existing instance of that signal type (data type information stored elsewhere)
- Re-Using same signal type: give it a name (not shown here)
- “Instantiation at design time” (as opposed to run-time)
Further narrowing of the gap

- **Transition** from requirements to logical architecture still complex
- **Two steps at the same time**
  - How to realize a feature?
  - How to realize distinct features using the same functions?

- **Also in the next product cycle:**
  - Requirements change
  - System won’t be developed from scratch
    - Re-Use
  - No **practically usable connection** between requirements for features and function net
    - Functional knowledge hidden in huge function net
General idea: feature views

- **Feature views** are a self-contained model of a feature
  - Document functional knowledge for later re-use
  - Traceable connection between feature models and logical architecture
  - Dependencies to other features modeled coarsely
  - Development teams know "their" features well, they can describe them completely
  - Includes variants, modes and scenarios
  - Not only software, but for better comprehensibility also hydraulics, cables, and other systems context, ...
How to model feature views?

- Use the same notation!
  - View = Feature function net
  - Similarity of notations helps to ease the transition
  - SysML supports using the same notation with different purposes
  - Separate tasks: Realize features as functions and embed view into whole vehicle function net

- View: May omit
  - Blocks, hierarchies, signals

- Can import blocks from other domains to clarify context
  - Add “environmental” blocks and non-signal communication to increase understandability
Simplified but illustrative example

In order to understand functional interrelations, additional information beyond software functions and signals is helpful

Function Nets: assumed to be more stable than software / hardware architecture (captures functional knowledge and omits technical details)

Function net describes a cyber physical feature.
Example feature function net

View describing feature “AutoLock”
- Left out blocks (e.g. CLRequestProc)
- Left out signals (e.g. StatusCL)
- Left out hierarchy (e.g. Doors)
- Included hardware (marked with stereotype “env”) and non-signal communication (marked with stereotype “M”)
- Included blocks from other features to model context

Complete function net
Consistency checks

- **Consistency conditions** needed between view and complete function net
  - Each block part of the complete net (apart from $\langle\langle\text{env}\rangle\rangle$)
  - Whole-part-relationships in the view present in the complete net (permitted to leave out intermediate layers)
  - Elements related via a (possibly transitive) whole-part-relationship in the complete net also in this relation in a view if both elements are shown
  - Communication relationships in the view present in the logical architecture (apart from non-digital).
  - Communication relationships need not be drawn to the exact source or target. Any superblock is sufficient if exact source or target block is omitted.

- View A on View B
  - Views A, B consistent with complete net and
  - A uses subset of elements in B

- Can be checked automatically to find inconsistencies
Using views for variants, modes and scenarios

- **Views on feature views**
  - Same consistency checks can be applied
  - In addition: Blocks and connectors are a subset of the elements in the feature view

- **Variants**
  - Valid configurations of features

- **Modes**
  - State-based model of different classes of behavior

- **Scenarios**
  - Exemplary behavior for simplified understanding and testing
  - UML Communication diagrams
Variants

- Automotive systems are **highly configurable**
  - Number of valid configurations may exceed number of produced cars

- Variant modeling with **feature trees**
  - Alternative features
  - Or-feature
  - Optional feature
  - Mandatory features

- Additional constraints limit the number of valid configurations
  - Some features require others
  - Some features exclude others
Vehicle features consists of
- sub-features + variability information

Standard way to model features
and variability: feature diagrams / feature trees
- We consider: Alternative / Or / Optional / Mandatory features
Connection to complete function net

- Complete function net is a “150%” model
- Feature tree models feature dependencies, exclusions
- **Challenge:** Comprehensive, traceable connection between feature model and function net
Connection to complete function net

- Established using views
  - Each feature in the tree has a view on its parent feature’s view
  - Unused blocks in union of sub-features: common functionality
- Relationship between feature and sub-feature(s) determines variability
Details: Alternative Features

**alternative features:**

Feature1

\[ \text{S1} \rightarrow \text{S2} \]

**fd**

**feature view:**

«view» ibd Feature1

\[ \text{A} \rightarrow \text{X} \]

\[ \text{I} \rightarrow \text{B} \]

\[ \text{Y} \rightarrow \text{C} \]

\[ \text{Z} \]

**ibd**

**(sub)feature views:**

«view» ibd S1

\[ \text{A} \rightarrow \text{X} \]

\[ \text{I} \rightarrow \text{B} \]

\[ \text{Y} \rightarrow \text{C} \]

«view» ibd S2

\[ \text{A} \rightarrow \text{X} \]

\[ \text{I} \rightarrow \text{B} \]

\[ \text{Y} \rightarrow \text{C} \]

\[ \text{Z} \]

**induced variant views:**

«view» ibd vS1

\[ \text{A} \rightarrow \text{X} \]

\[ \text{I} \rightarrow \text{B} \]

\[ \text{Y} \rightarrow \text{C} \]

«view» ibd vS2

\[ \text{A} \rightarrow \text{X} \]

\[ \text{I} \rightarrow \text{B} \]

\[ \text{Y} \rightarrow \text{C} \]

\[ \text{Z} \]
mandatory and optional features:

(sub)feature views:

induced variant views:
Details: Or-Features

or-features:

- Feature2
  - S1
  - S2

feature view:

- «view» ibd Feature2
  - A
    - X
  - B
    - Y

(sub)feature views:

- «view» ibd S1
  - A
    - X
  - B
    - Y

induced variant views:

- «view» ibd vS1
  - A
    - X
  - B
    - Y

- «view» ibd vS2
  - A
    - X
  - B
    - Y

- «view» ibd vS1S2
  - A
    - X
  - B
    - Y
Methodological Aspects

- Approach
  - **extends** (but not replace) methodology and tool chain
    - Establishes connection from feature requirements and their variability to logical architecture as function net
  - **reuses** large parts of logical architecture and views is key requirement

- Development steps for logical architecture
  - Reuse complete logical architecture and views from an earlier vehicle generation
  - Adapt feature diagrams and views to new requirements
  - Develop self-contained views for new features
  - Early and repeatedly execute consistency checks between views and complete function net to identify inconsistencies or unused blocks and resolve manually
Modes

- What is a mode?
  - System shows **substantial different behavior** in different modes
  - Modes are not frequently changed in a running system
- Modes are typically used to model
  - **error degradation**
  - different **modes of operation**
- Modes are described by two types of diagrams
  - Statechart
    - States refer to modes of the feature
    - Transitions describes the condition under which different modes are switched
  - Mode function nets
    - Mode function net is a view on the feature function net or complete system
Using Views to Model Error Degradation

Complete function net

Only communicating blocks are shown
Conclusion

- **Logical architecture as function nets** as a smoothing step in development process

- Based on a **SysML notation**: Internal block diagrams

- Views use **same notation** as function nets and are suitable for modeling **features, variants, modes and scenarios** in a similar way

- Provided **consistency conditions** between views and the function net

- Smaller experiments promising, but
  - Adding another layer of modeling may increase overhead
  - Larger case studies needed to rate the effort and acceptance by developers
  - Tool support necessary