Westsächsische Hochschule Zwickau (FH)
University of Applied Sciences

Concepts for Safety-Inherent Model-Driven Software Family Engineering and Product Configuration in the Automotive Controller Software Domain

Frank Grimm, Westsächsische Hochschule Zwickau (FH)

Automotive 2006, Stuttgart, 12th October 2006

Outline

Introduction
- OMOS Software Development Process
- OMOS Modelling Approach for Software Families

Product Configuration Problem

Problem Solution
- Metamodel-based Approach
- Formal Modeling Rules
- Tool Support

OMOS Modelling Approach for Software Families

- OMOS = object-oriented modelling of software in the ECU domain
- Bosch uses OMOS to create ECU software for automatic gearboxes
- OMOS is a visual, model-driven technique
- UML class models used to model the architecture ECU software systems

Software Product Families

- Set of software systems sharing a common set of features that satisfy specific needs of a particular market segment

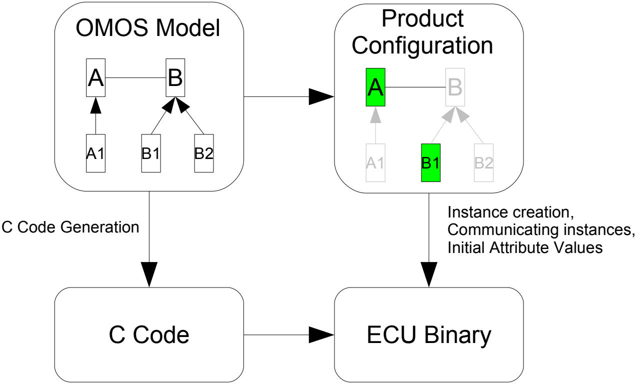
→ Various product variants can be derived from the basic product family

Need for product family

- Gearbox software is developed for 5 different manufacturers
- Large diversity of customer requirements

→ Delivered systems are different in the implementation, but share common functionalities and architecture

OMOS Software Development Process



Creating Software Product Families with OMOS

- Base classes represent functionalities of an ECU software system

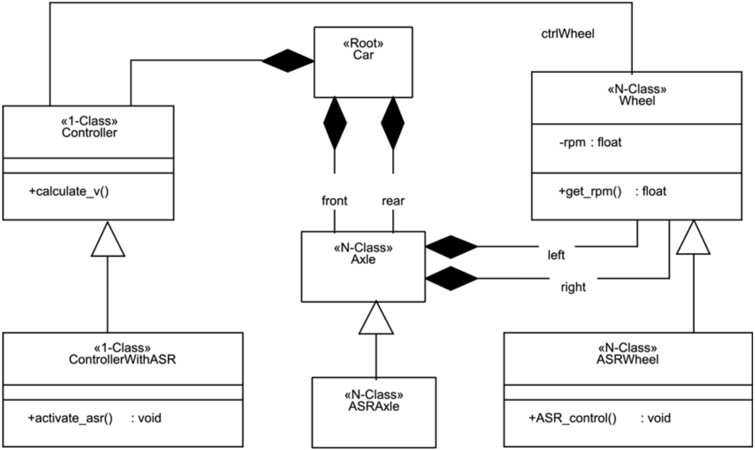
→ Base class introduces functionality (variation point)

- Sub-classes represent variations of particular functionality

→ Sub-classes used to realize requirements of different customers on the same functionality

Software Product Families

Example



Software Product Configuration

- OMOS model contains all variants

→ All products are based on the same model

- Product configuration = selecting the proper variants that fulfil customer's requirements of a specific project

Anti Slipping Regulation (ASR) – Example

- Variants: `ASRAxle`, `ASRWheel` and `ControllerWithASR`

Product Configuration Problem

- Analysed sub-system: 300 classes, 100 functionalities (variation points), 2 to 5 variants per variation point
- Selecting proper combination of variants for a certain product is error-prone
- Knowledge about dependent variants is currently not explicitly included in models

→ Dependency solving solely based on the knowledge of software engineers who have to be aware of implicit dependencies between variants

Types of Configuration Errors

- Dependencies between variations that are not directly related
  e. g., `ASRAxle` requires `ControllerWithASR`

→ Behaviour of particular variant implicitly depends on other variant's behaviour

- Variations that are directly related
  e. g., `Controller` explicitly depends on `Wheel`

→ Selecting wrong sub-class cannot be prevented
  e. g., `ControllerWithASR` requires `ASRWheel`

- Majority of errors results from combining variants which implement different behaviour than the required variants

→ Result: undefined run-time behaviour

Product Configuration Problem

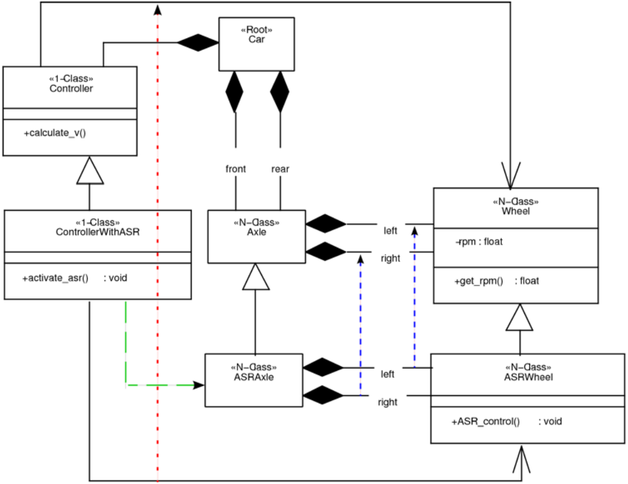
- Guarantee that delivered product fulfils customer's requirements
- Reliable product configuration process

$\longrightarrow$ Reducing ambiguity during configuration

$\longrightarrow$ Restricting the combination of variants using explicit dependencies

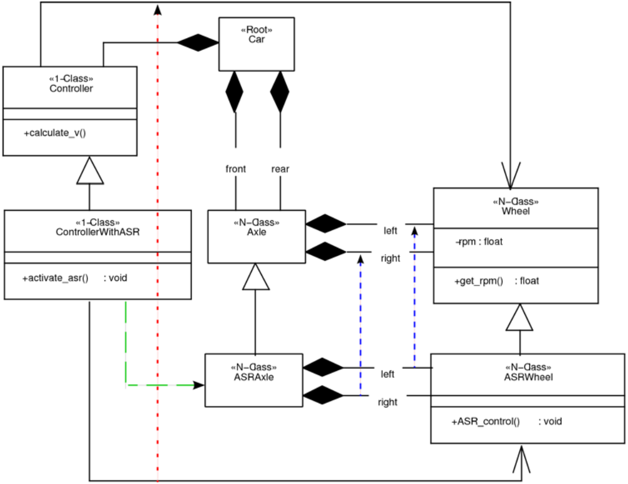
Solving the Configuration Problems

- Variants refine inherited aggregations and associations

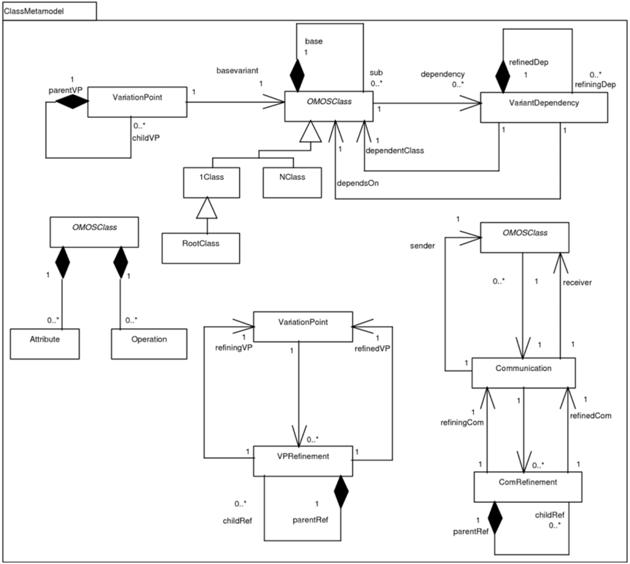


Solving the Configuration Problems

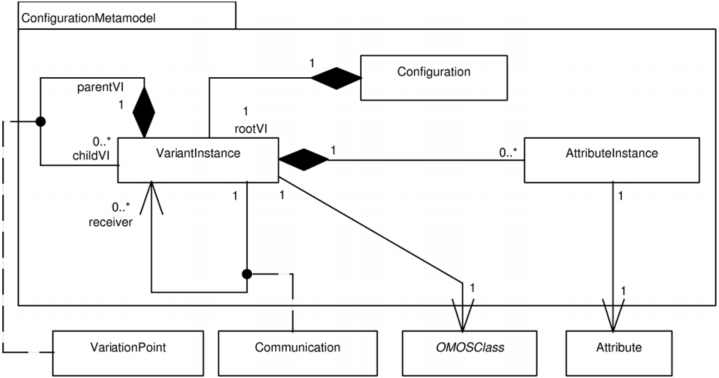
- Implicitly related variations become explicit



Metamodel for OMOS Models



Configuration Metamodel



Domain-specific Metamodelling Approach

- Domain-specific metamodels
- Modelling rules
- Tool support

Advantages

- Metamodel describes concepts of ECU software engineering domain
  $\rightarrow$ understood by domain experts
- Both metamodels are considerably smaller than UML metamodel
- Mapping between UML metamodel of conventional UML CASE tools and domain-metamodel possible

Defining formal Modelling Rules

- Rules describe, constrain and verify usage of model elements
- Rules are based on domain-specific metamodel elements
- Common Object Constraint Language (OCL) used to describe rules

Tool Support

- In-place checking
  - Include meta-models and rules engine into CASE tool
  - → Errors can be detected early during the modelling phase
- External checking
  - Extract model information and verify models (before product configuration) and configurations
  - → Checker is independent of CASE tool

Conclusion

- Product configuration problem of software families
- Metamodel-based solution allows for explicit modelling and management of dependencies between variants
- Modelling rules for reliable configurations
- Tools to verify rules

Thank you for your attention!