# Generative Programming

## Methods, Techniques, and Applications










Ulrich W. Eisenecker
University of Applied Sciences
Kaiserslautern, Zweibrücken
*eisenecker@informatik.fh-kl.de*

Krzysztof Czarnecki &
Simon Helsen
University of Waterloo
*czarnecki@acm.org*
*shelsen@computer.org*

# Goals

- What is to be achieved?
  - Basic understanding of Generative Programming (GP)
  - Based on first impressions and discussion you should be able to judge what you can benefit from GP
- How can it be achieved?
  - Overview of the most important steps of GP
  - Demonstration of tools
  - Presentation of case studies
  - Discussion in the plenum

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview - Parts

- Introduction and Example
- Feature Modeling
- Frames
- MDA
- Questions and Discussion

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Generative Programming

## Methods, Techniques, and Applications










Introduction and Example

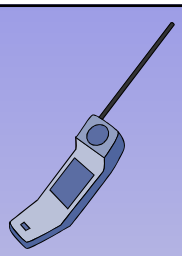
# Overview

- ➢ Challenges
- Introduction
- First Example
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



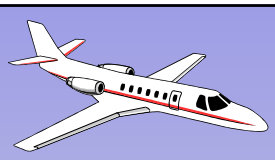
# Complexity



- Software systems are engineering in new and demanding application domains
- Software systems support increasingly complex tasks
- Requirements concerning functionality and efficiency are constantly growing

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Quality

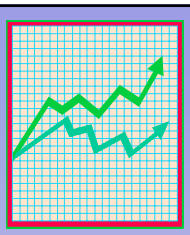


- Software is deployed in life-critical application areas
- Software malfunction causes economic losses
- Usability is a major acceptance criterion
- Users expect a specific quality that is determined by varying criteria

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



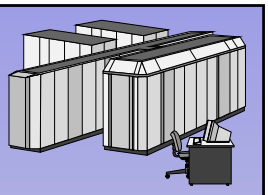
# Productivity



- Growing demand for new systems and variants
- Increase of productivity and development staff cannot close the gap between demand and supply
- Standard software becomes more and more important, but it cannot be adapted to every specific requirement

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Maintenance & Legacy Systems



- Long lifetime of software systems (sometimes 20 years and more)
- Software must be continuously adapted to new and changing requirements
- Maintenance of legacy systems absorbs development capacity to a high degree
- Software systems of today are the legacy of tomorrow!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Example "Object Technology"

- Theory
  - Productivity and quality should increase thanks to reuse
- Reality
  - Classes are too small as units of reuse
  - Frameworks are sufficiently large as units of reuse, but frameworks from different vendors do not integrate well
  - Design patterns are pieces of reusable knowledge, but they do not exist as executable code

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Example "Component Technology"

- Standards and platforms improve interoperability and the distributed execution of components
- Many components provide a special mode for customization
- Reusing small components does not have a large impact on software development while large components require high customization efforts

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# One of the Causes ...

Currently dominating methods and tools are designed for developing ***single systems***

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Challenges
- ➢ Introduction
- First Example
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Single System Development

- Focus on analyzing, modeling, and implementing a single system
- Important goal: decomposition into components, i.e. modularization
  - Individual components are easier to maintain, to replace, and to reuse
  - But, what are the criteria for decomposition?

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Experiment - Part I

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Experiment - Part II

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Exemplar Systems of a Family

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# The "Right" Modules

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# An Additional System …

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# … Can Cause Surprise!

This page is intentionally left blank!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# How To Order Systems?

- By Components
  - Downward arrow on top
  - Large triangle below with its 90° angle at the lower left point
  - …





- By Specification
  - A cat
  - Sitting upright
  - Waiving its tail



© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# From Order To Product



Creating the order
- Expert language
- Form, catalogue
- By example



Processing the order
- Completion
- Validation
- Construction rules
- Optimization





Product generation
- Assembling elementary, reusable components

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Generative Programming …

... is a software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

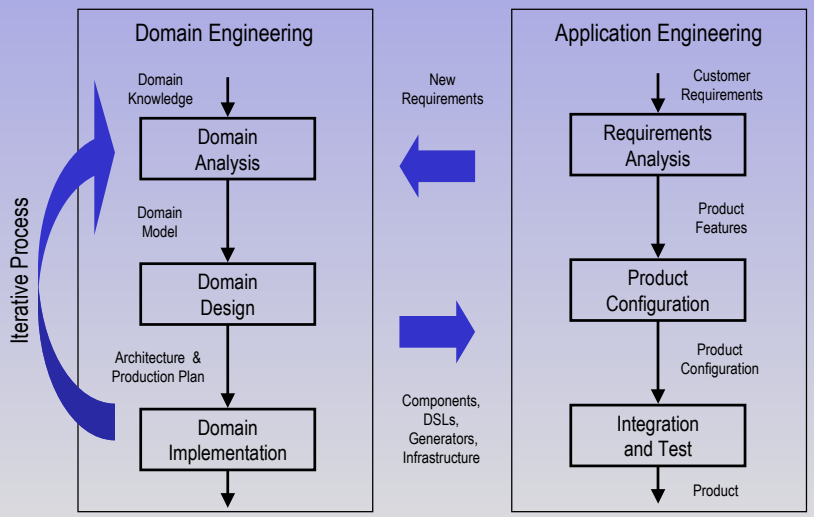


# Two Processes

- Domain Engineering
  - Analysis: Domain scoping and defining a set of reusable, configurable requirements for the systems in the domain
  - Design: Developing a common architecture for the systems in the domain and devising a production plan
  - Implementation: Implementing the reusable assets, for example, reusable components, domain-specific languages, generators, a reuse infrastructure, and a production process
- Application Engineering
  - Producing concrete systems using the reusable assets developed during Domain Engineering.

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# System Family Approach

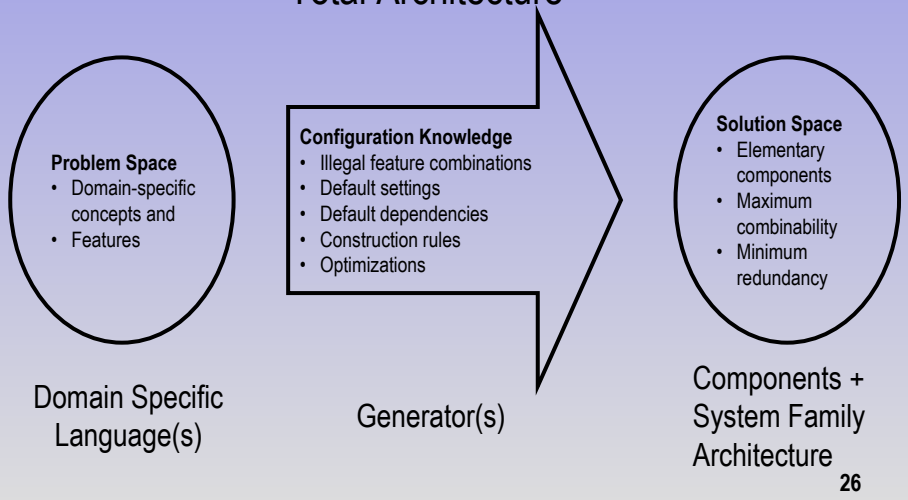



© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



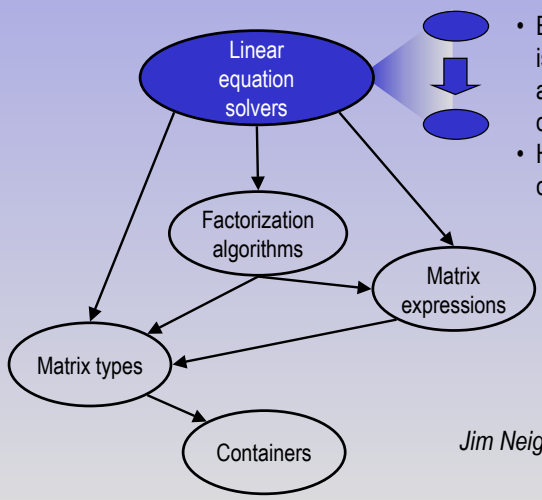
# Generative Domain Model

## Total Architecture




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Domain Network



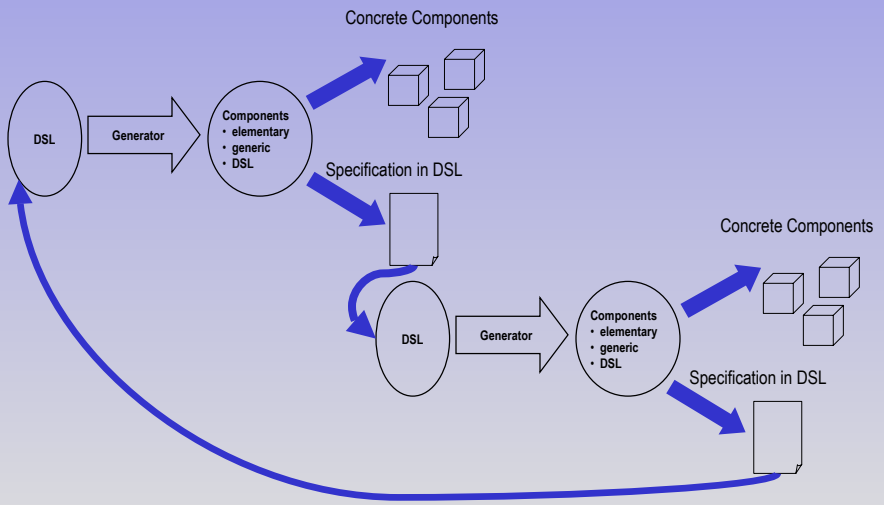

- Each subdomain is implemented as a generative domain model
- Higher order components

*Jim Neighbors, Draco, 1980*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Federated Generators




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# A Technology Projection ...

… is a mapping of the generative domain model onto

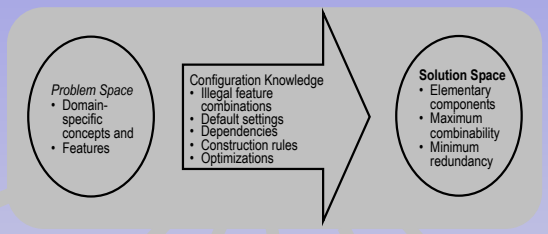
- other software development paradigms,
- a programming language,
- several development tools that are combined within one environment or on one platform

- Meanwhile, several technology projections are available

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Existing Technology Projections




**C++**
- Template meta-programming
- Static configuration
- Language-specific means only

**Java and JavaBeans**
- Interactive parameterization of JavaBeans
- Dynamic configuration
- Language-specific means only

**Java und AspectJ**
- Using an AspectWeaver
- Static configuration
- Separate preprocessing step

**Not a mapping of one specification onto many platforms!**

**ANGIE**
- Employment of a frame processor
- Static configuration
- Separate preprocessing step

**Small Components**
- Usage of an XML configuration file
- Static and dynamic configuration
- Separate preprocessing step

**XML, TL and Ada 83**
- Graphically-interactive configuration
- Static configuration
- Separate preprocessing step

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Essential Terms

**In this presentation**

- Domain Analysis is part of Domain Engineering
- A product family comprises the variants of a product
- A product line comprises multiple products that are targeted for a specific group of customers
- A product line can be based on system families

**Different uses**

- Domain Analysis precedes Domain Engineering
- Product family and product line are synonym

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Essential Terms

- Generic
  - *"relating to or characteristic of a whole group or class"* (Merriam-Webster Online)
  - Solution space technique for developing parameterizable components
- Generative
  - *"having the power or function of generating, originating, producing, or reproducing"* (Merriam-Webster Online)
  - System for producing other systems; it comprises problem space, configuration knowledge, and solution space

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Scope

- Generative programming is not limited to implementing application code, testing, and creating GUIs
- System family engineering also includes
  - documentation
  - system installation
  - user training
  - maintenance, and many more

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

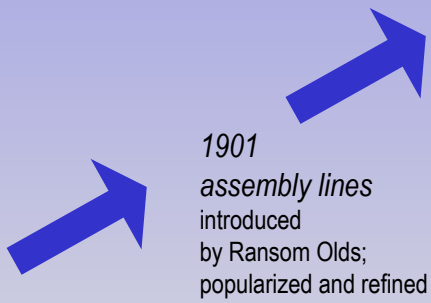


# Overview

- Challenges
- Introduction
- ➢ First Example
  - Feature Model
  - Implementation Components
  - System Family Architecture
  - Domain Specific Language
  - Generator
  - Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Industrial Revolution



*1826*
*interchangeable parts*
successfully introduced by John Hall (after 25 years of unsuccessful attempts!)

*1901*
*assembly lines*
introduced by Ransom Olds; popularized and refined by Henry Ford in 1913

*1980s*
*automated assembly lines*
first industrial robot installed in 1961 at General Motors; 1970's advance of microchips

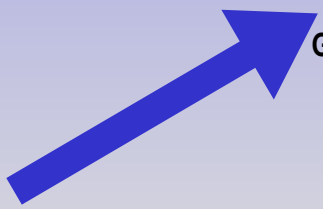
© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Development with Components

Generating the needed components and automatic assembly

**Generative Programming**



**Current Practice**

Searching for and adapting components and manual assembly

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# The Programmer States ...

... *what* she needs

CAR_GENERATOR<*EcoLine,TrailerCoupling*>::**RET**
LIST_GENERATOR<*Base,cp,poly,with_counter*>::**RET**
ACCOUT_GENERATOR<*FixedTermDeposit*<>,*EUR*>::**RET**
MATRIX_GENERATOR<*matrix*<*double,structure*
<*rect*<> > > >::**RET**

and gets the **desired component**!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Overview

- Challenges
- Introduction
- First Example
  - ➢ Feature Model
  - – Implementation Components
  - – System Family Architecture
  - – Domain Specific Language
  - – Generator
  - – Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

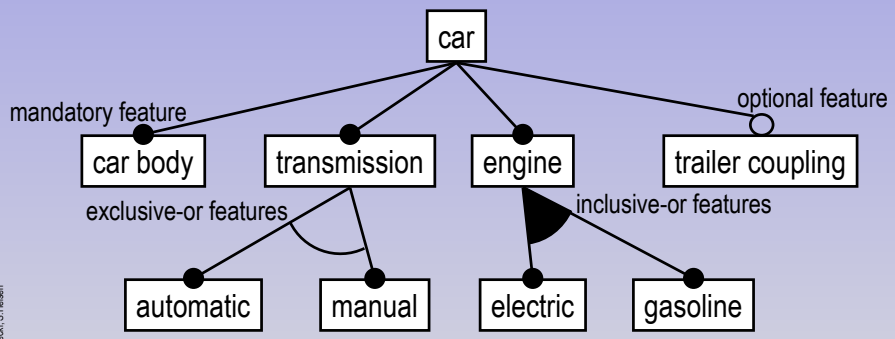
# Domain Engineering: Analysis

- Stakeholder analysis
- Domain scoping
- Analysis of common and variable features and their dependencies
- Documentation using feature models (FODA)
  - Feature diagrams representing commonalities and variabilities in an implementation-independent and easy to understand manner
  - Tables and free text documenting feature dependencies, short descriptions, origin, rationale, defaults, binding time, usage examples, etc.

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



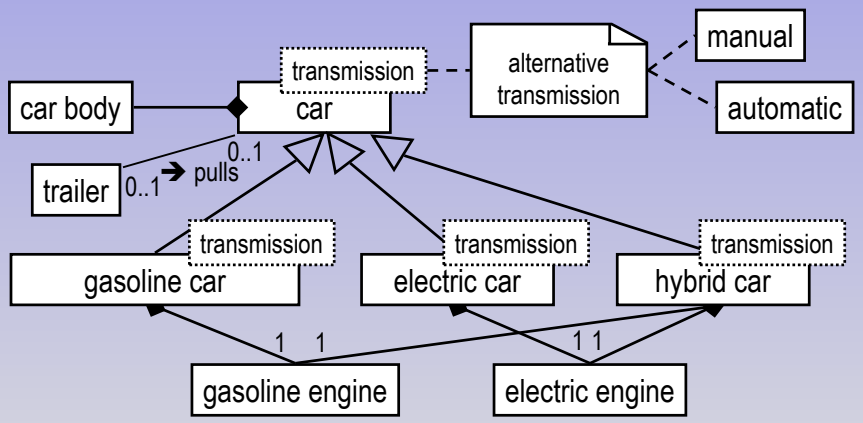
# Feature Diagram




No decision regarding the mechanism for implementing variability!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# To Compare: A UML Class Diagram




Variability is implemented using concrete mechanisms, e.g. inheritance, aggregation, associations, and parameterization

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Feature Combinations

- The feature diagram contains
  2 (transmission)
  x 3 (engine)
  x 2 (trailer coupling)
  = 12 combinations
- Constraint (to be documented separately): electric engine requires automatic transmission

➡ i.e., we have 8 valid combinations

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Challenges
- Introduction
- First Example
  - Feature Model
  - ➢ Implementation Components
  - System Family Architecture
  - Domain Specific Language
  - Generator
  - Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Domain Engineering: Design

- Identifying implementation components
- Grouping implementation components by equivalence categories
- Dependency analysis
- Architecture for the system family
  - GenVoca layered architecture
  - GenVoca grammar

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

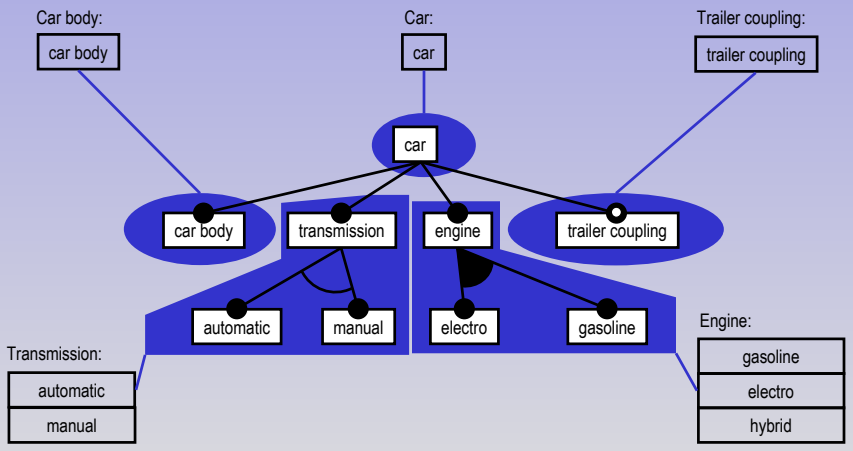
# Equivalence Categories

- What constitutes an equivalence category?
  - Features that are mapped to concrete components
  - Single mandatory leave
  - Single optional leave
  - Inclusive-or group
  - Exclusive-or group
- Dependency analysis is performed on implementation components
- Results of dependency analysis are depicted on the level of equivalence categories

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

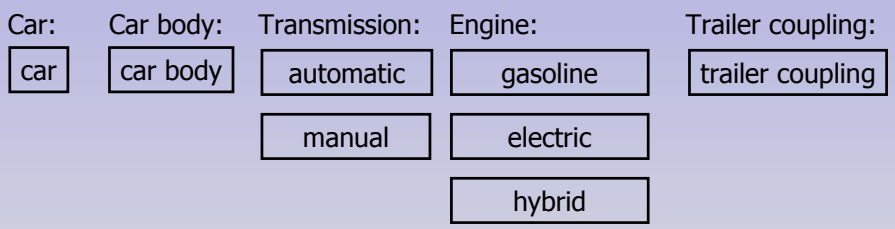


# Equivalence Categories




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Implementation Components




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

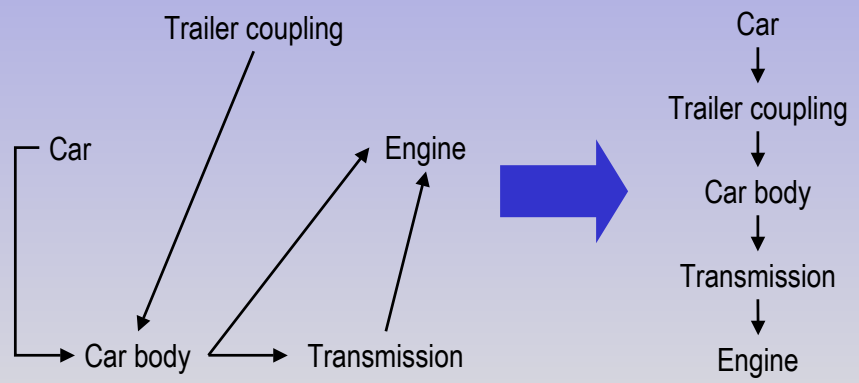


# Dependency Analysis

- Dependency analysis builds on implementation components grouped in equivalence categories
- Implementation components do not need to be (fully) implemented for dependency analysis
- Identification of dependencies among implementation components
- Provides the basis for defining a common architecture for all systems of a family

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Example of Dependency Analysis




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

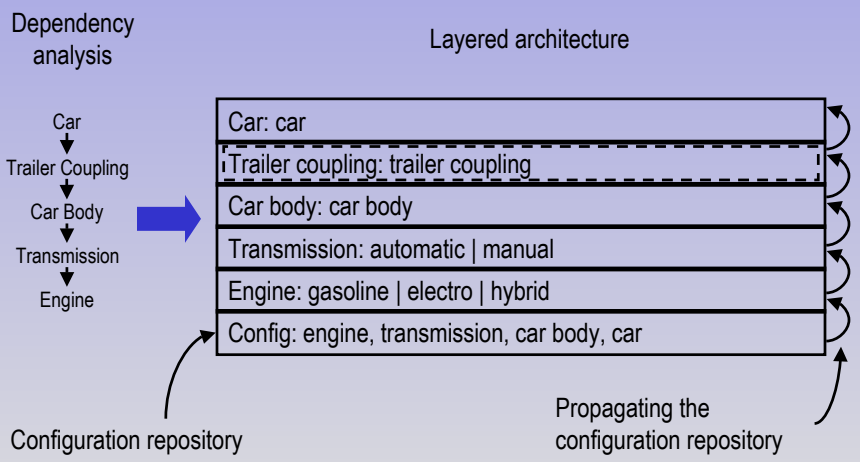


# Overview

- Challenges
- Introduction
- First Example
  - Feature Model
  - Implementation Components
  - ➢ System Family Architecture
  - Domain Specific Language
  - Generator
  - Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# GenVoca Layered Architecture




Home Page of the GenVoca Project,
*www.cs.utexas.edu/users/schwartz*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# GenVoca Grammar

| | |
|---|---|
| Car: | Car[CarBodyWithOptTC] |
| CarBodyWithOptTC: | TrailerCoupling[CompleteCarBody] \| CompleteCarBody |
| CompleteCarBody: | CarBody[TransmissionWithEngine] |
| TransmissionWithEngine: | ManualTransmission[Engine] \| AutomaticTransmission[Engine] |
| Engine: | GasolineEngine[Config] \| ElectricEngine[Config] \| HybridEngine[Config] |

---

| | |
|---|---|
| Config | |
| Engine: | GasolineEngine \| ElectricEngine \| HybridEngine |
| Transmission: | ManualTransmission[Engine] \| AutomaticTransmission[Engine] |
| CarBody: | CarBody[TransmissionWithEngine] |
| ... | |

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Car Components

```
template <class Config_>
struct GasolineEngine {
  typedef Config_ Config;
 GasolineEngine () {
    cout << " GasolineEngine ";
  }
};

template <class Engine>
struct AutomaticTransmission: Engine {
  typedef typename
         Engine::Config Config;
 AutomaticTransmission() {
    cout << " AutomaticTransmission ";
  }
};
```

```
template <class CarBody _>
struct TrailerCoupling: CarBody_ {
  typedef typename
         CarBody _::Config Config;
 TrailerCoupling() {
    cout << " TrailerCoupling ";
  }
};

template <class CarBodyWithOptTC>
struct Car: CarBodyWithOptTC {
  typedef typename
         CarBodyWithOptTC ::Config Config;
  Car() { cout << "Car " << endl; }
};
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Configuration

```
struct C1 {
       typedef ElectroEngine<C1> Engine;
       typedef AutomaticTransmission<Engine> Transmission;
       typedef CarBody<Transmission> CarBody;
       typedef Car< CarBody > Car;
     };
// …
struct C12 {
       typedef HybridEngine<C12> Engine;
       typedef ManualTransmission<Engine> Transmission;
       typedef CarBody<Transmission> CarBody;
       typedef TrailerCoupling<CarBody> TrailerCoupling;
       typedef Car< TrailerCoupling > Car;
};
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Manual Configuration

- Manually writing 12 configuration repositories is tedious and error prone
- Only 8 configuration repositories are valid
- How to automatically produce valid configurations?

➡ Configuration generators

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Overview

- Challenges
- Introduction
- First Example
  - Feature Model
  - Implementation Components
  - System Family Architecture
  - ➢ Domain Specific Language
  - Generator
  - Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Design Criteria for DSLs

- Usability
  - Simplicity: easy to understand, to learn, and to use
  - Efficiency: only a minimal set of language elements should be required to order a maximal number of variants (provide default values for omitted features)
  - Accuracy: minimal probability for ordering an invalid system
  - Openness: new language elements should be added with minimal effort
- Implementation
  - Modular structure
  - easily extensible
  - easy to analyze

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Inspirations for DSL-Design

- Feature diagram
  - Omit features which do not need to be specified
  - Transforming a group of optional features into a dimension, thus avoiding invalid specifications
  - Reorder features being more frequently specified so that they occur before feature which are less frequently mentioned
  - Introducing additional abstract features
- Borrow from existing DSLs, for example catalogues for ordering cars
- Involve users and experts when designing innovative DSLs

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Ordering Cars

According to the catalog, ordering invalid configurations is not possible, e.g. electric engine with manual transmission

| | BaseLine | CityLine | EcoLine |
|---|---|---|---|
| **Engine** | | | |
| gasoline | ● | | |
| electric | | ● | |
| hybrid | | | ● |
| **Transmission** | | | |
| five-speed manual | ● | | |
| four-speed automatic | ❍ | ● | ● |
| **Options** | | | |
| trailer coupling | ❍ | ❍ | ❍ |

● standard equipment
❍ options available for a surcharge

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Domain Specific Language

```
// implementation of the catalog in a domain-specific language (DSL)
enum Transmission { fiveSpeed, fourSpeedAutomatic };
enum Line { baseline, cityline, ecoline }; // implementation detail

template <int transmission_ = fiveSpeed>
struct BaseLine {
    enum { transmission = transmission_,
           line = baseline
         };
};
// ...
enum Options { none, trailerCoupling };
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Challenges
- Introduction
- First Example
  - Feature Model
  - Implementation Components
  - System Family Architecture
  - Domain Specific Language
    - Generator
  - Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Configuration Generator

```
            // Remaining part of DSL
template <class Type = BaseLine<>,int options_ = none>
struct CAR_GENERATOR {
    // Define a shorthand for CAR_GENERATOR<Type,options_>
    typedef CAR_GENERATOR<Type,options_> Generator;
    // Analyze specification in DSL
    enum    {   model_         = Type::model_,
                transmission_  = Type::transmission_
            };

// Constructibility not checked; DSL gives no chance to make a mistake
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Configuration Generator

```
typedef SWITCH<line_,
        CASE<baseline,GasolineEngine<Generator>,
        CASE<cityline,ElectricEngine<Generator>,
        CASE<ecoline,HybridEngine<Generator> > > > >::RET Engine_;
typedef IF<(transmission_ == fiveSpeed),
        ManualTransmission<Engine_>,
        AutomaticTransmission<Engine_> >::RET Transmission_;
typedef IF<(options_ == trailerCoupling),
        TrailerCoupling<CarBody<Transmission_> >,
        CarBody<Transmission_> >::RET CarBody_;
typedef Car<CarBody_> RET; // Complete car type
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Template Metaprogramming

- C++ templates define a Turing-complete, functional language executed at compile time (➡Erwin Unruh, Todd Veldhuizen)
- C-like control structures IF, SWITCH, DO, WHILE, FOR as well as EVAL_TABLE available
- Czarnecki, Eisenecker: Generative Programming - Methods, Techniques, and Applications. Addison Wesley, 2000

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Configuration Generator

```
// Define the configuration repository

    struct Config {
      enum { model        = model_,
             transmission = transmission_,
             options      = options_,
      };
      typedef RET Car;
    };
// The configuration repository is a part of each product; i.e., it can be
// accessed by other generators (e.g. generating matching appliances)
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# A Small Change is Still Needed...

```
template<class Generator> // Generator instead of Config
struct GasolineEngine
{ typedef typename Generator::Config Config;
  GasolineEngine() { cout << "GasolineEngine "; }
};

// ElectricEngine
...

// HybridEngine
...
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Usage

```
typedef CAR_GENERATOR<>::RET C1; // Give me a default one
typedef CAR_GENERATOR<BaseLine<>,TrailerCoupling>::RET C2;
typedef CAR_GENERATOR<BaseLine<fourSpeedAutomatic> >::RET C3;

// ...

C1 c1; C2 c2; C3 c3;

// ...
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

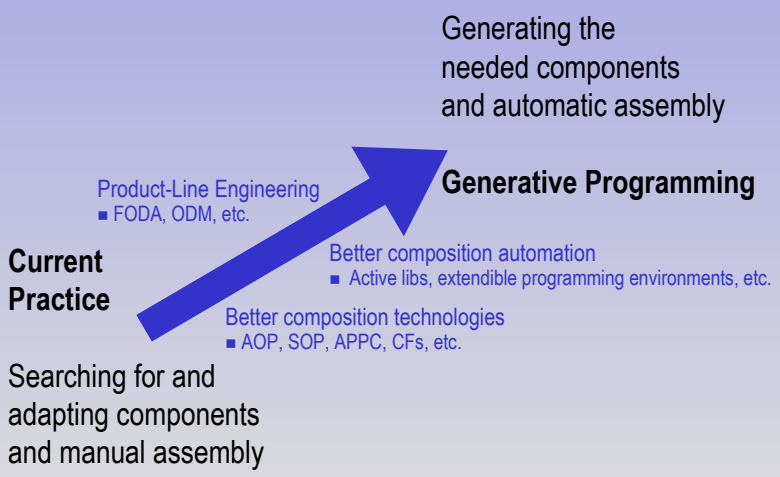


# Overview

- Challenges
- Introduction
- First Example
  - Feature Model
  - Implementation Components
  - System Family Architecture
  - Domain Specific Language
  - Generator
  - ➢ Prospects
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Industrializing Software Production




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Theses

- A component is always a part of a well-defined production process
- If you can assemble the components manually, you can also automate it using a generator
- Generation can be performed both statically and dynamically
- Criteria such as binary format, interoperability, language independence, etc., are always relative to the production process

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Challenges
- Introduction
- First Example
- ➢ Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# References


- Bosch: Design and Use of Software Architecture: Adopting and evolving a product-line approach. Addison-Wesley, 2000
- Cleaveland: Program Generators with XML and Java. Prentice-Hall, 2001
- Clements, Northrop: Software Product Lines: Practices and Patterns. Addison-Wesley, 2001


© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# References


- Czarnecki, Eisenecker: Generative Programming. Methods, Tools, and Applications, Addison-Wesley, 2000
- Donohoe (Hg.): Software Product Lines: Experience and Research Directions, Proceedings. Kluwer Academic Publishers, 2000
- Weiss, Lai: Software Product-Line Engineering. A Familiy-Based Software Development Process. Addison-Wesley, 1999


© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Web References

- *http://www.codegeneration.net*
- *www.aisto.com/roeder/Paper/*
- *www.delta-software-technology.com*
- *www.generative-programming.org*
- *www.gp-web.org*
- *www.iese.fhg.de*
- *www.omniscium.com/?page=IntentionalFaq*
- *www.polite-project.org*
- *www.program-transformation.org*
- *research.sun.com/projects/jackpot/*
- *www.sei.cmu.edu/plp/framework.html*
- *www.sei.cmu.edu/programs/pls/*
- *www.softwaregenerators.com*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Practice

- On demand
  - Examination of the program for manual composition of cars (car_cfg.cpp)
  - Examination of the program for generating cars (car_gen.cpp)
  - Extending the car generator

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Generative Programming

## Methods, Techniques, and Applications

Fachhochschule Kaiserslautern University of Applied Sciences

THINK GLOBAL fh ZWEIBRÜCKEN CAMPUS

University of Waterloo

Feature Modeling

# Overview

- ➢ Elements and Notation
- AmiEddi-XML 1.3
- Captain Feature
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Feature Modeling

- Feature modeling is about identifying
  - common and
  - variable features of concepts and their
  - dependencies

  and documenting them in a coherent model, called a feature model
- Feature modeling is a core activity of important domain-engineering methods, such as ODM and FODA
- Features help to identify implementation components and designing DSLs

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Concepts and Classes

- Aren't these equations generally true?
  - Concept = class
  - Object = instance of a concept
- No!
  - Objects have predefined semantic properties such as identity, state, behavior
  - Instances of concepts do not have any predefined semantics, they could be virtually anything
  - There are so many variants of instances and concepts that they can't be described using a single class only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# A Feature Model Consists of

- Feature diagram(s) and
- additional information
  - Short semantic description of each feature
  - Rationale for the relevance of each feature
  - Stakeholders and systems that are interested in a feature
  - Examples of systems with that feature
  - Restrictions
  - Dependencies and default values
  - Availability (where, when, and to whom is the feature available?)
  - Binding (where, when, and who binds the feature?)
  - Binding mode (dynamic, static)
  - Open/closed (is it possible to add sub-features later?)
  - Priority (how important is the feature?)
  - ...

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Feature Diagram

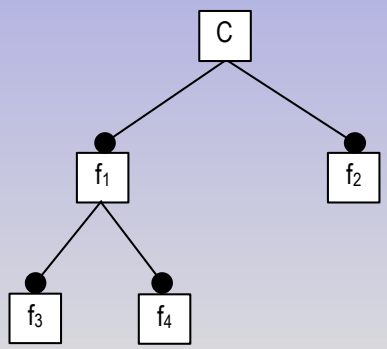



- The root node C represents the concept C
- All other nodes are features
- f1 is a direct feature of C
- f2 and f3 are indirect features of C
- f2 is a direct subfeature of f1
- f3 is an indirect subfeature of f1

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



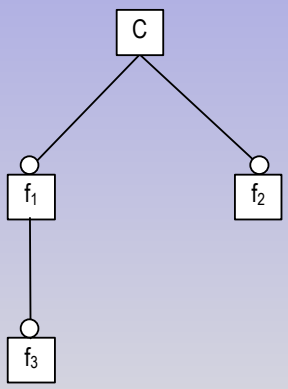
# Mandatory Feature




- A mandatory feature is part of a concept instance description only if its parent is also part of the description
- Mandatory features are pointed to by edges with a filled circle, e.g. f1,f2,f3, and f4
- All instances of C are described by the feature set {C,f1,f2,f3,f4}

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Optional Feature

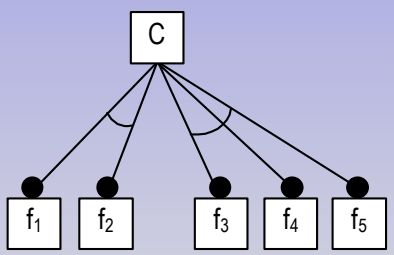



- An optional feature can be part of a concept instance description only if the parent node is also part of the description
- Optional features are pointed to by edges with an empty circle (e.g., f1,f2, and f3)
- The following sets describe instances of C: {C}, {C,f1}, {C,f1,f3}, {C,f2}, {C,f1,f2}, {C,f1,f3,f2}

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



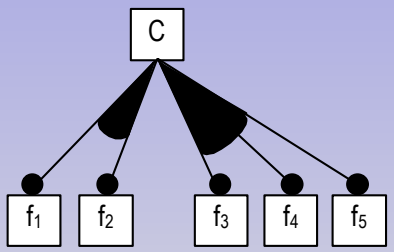
# Exclusive-Or Features




- Exactly one from a set of exclusive-or features is part of a concept instance description if its parent node is also part of the description
- Edges pointing to exclusive-or features of one set are connected by an empty arc
- The following sets describe instances of C: {C,f1,f3}, {C,f1,f4}, {C,f1,f5}, {C,f2,f3}, {C,f2,f4}, {C,f2,f5}

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Inclusive-Or Features

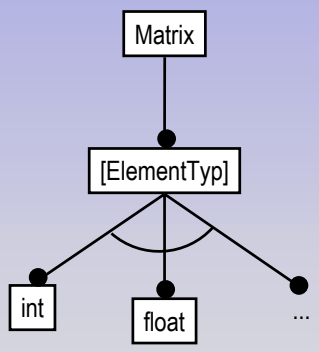



- Any non-empty subset from a set of optional features can be part of a concept instance description if the parent node is also part of it
- Edges pointing to inclusive-or features of one set are connected by a filled arc
- The diagram denotes 21 different concept instances

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Open/Closed Features




- An open feature is expected to be refined with further subfeatures
- In a feature diagram, brackets [] are used to indicate openness
- We can also show selected examples of subfeatures (not part of the formal notation)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

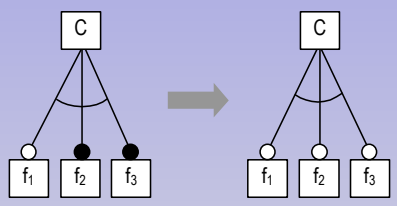
# Normalized Feature Diagrams

- Besides mandatory, optional, and inclusive-or features, there are also
  - optional exclusive-or features and
  - optional inclusive-or features
- This results in redundant representations
- Each feature diagram can be transformed such
  - that it does not contain any optional inclusive-or features
  - and any set of exclusive-or features contains either exclusive-or or optional exclusive-or features only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



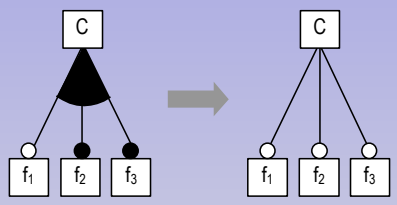
# Normalized Feature Diagrams




- If one or more of the features in a set of exclusive-or features is optional, it has the same effect as if all the exclusive-or features in this set were optional

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Normalized Feature Diagrams




- If at least one or more of the features in a set of incluive-or features is optional, it has the same effect as if all the features in this set were optional features

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Commonality in Feature Diagrams

- From the perspective of a concept
  - What are the common features of all concept instances?

  ➡Common Features
- From the perspective of a feature
  - What are the common features of all concept instances that have the special feature?

  ➡Common Subfeatures

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

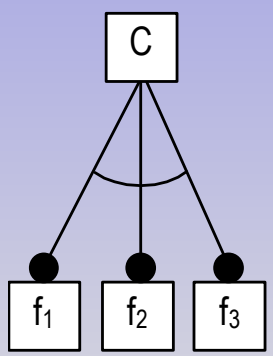
# Variability in Feature Diagrams

- Variable features describe variability
  - optional, exclusive-or, optional-alternative, and inclusive-or features
- The nodes having variable subfeatures are so-called variation points (VP)
  - homogeneous vs. inhomogeneous VPs: all direct subfeatures are of the same vs. different kinds
  - singular vs. nonsingular VPs: none or one vs. many direct subfeatures can be selected

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Kinds of Variation Points

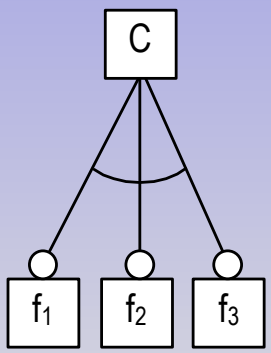



Dimension

- Feature or concept with all direct subfeatures or features being exclusive-or features only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Kinds of Variation Points



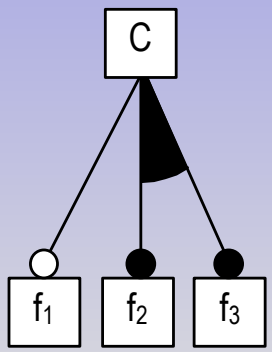

Dimension with optional features

- Feature or concept with direct subfeatures or features being exclusive-or-optional features only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Kinds of Variation Points

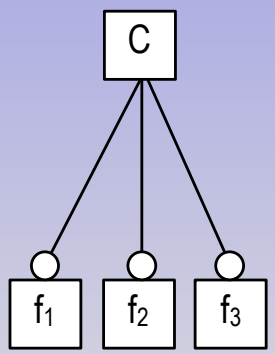



Extension point

- Concept or feature with at least one set of direct inclusive-or features or inclusive-or subfeature
- Concept or feature with at least one direct optional feature or subfeature

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Kinds of Variation Points



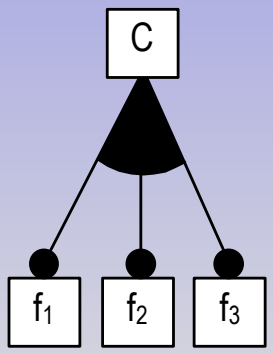

Extension point with optional features

- Concept or feature with all direct features or subfeatures being optional features only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Kinds of Variation Points




Extension point with inclusive-or features

- Concept or feature with all direct features or subfeatures being inclusive-or features only

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# FAQs

*What are the semantics of the edges of a feature diagram?*

- Edges don‘t have their own full semantics
- The decoration of the edges completes the semantic: a certain feature must, can, or can not be part of a concept instance description
- Edges don‘t represent part-of or inheritance relationships!

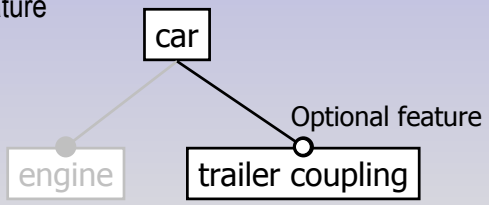
© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# FAQs

*How to describe the configurability aspect of a relation if edges don‘t possess relationship semantics?*

- If a feature diagram contains a variable relation it should be modeled as a feature
- For example, the option ”trailer coupling" is represented as an optional feature




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# FAQs

*Isn't a feature diagram just a simple part-of hierarchy?*

- No, features can represent part-of relations, but they can carry any other meaning too
- Features can be implemented quite differently, e.g.
  - concrete features as concrete components
  - aspect features, e.g. synchronization, are often woven into other components
  - abstract features, such as memory usage optimization, determine the configuration of other components

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Overview

- Elements and Notation
- ➢ AmiEddi-XML 1.3
- Captain Feature
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# AmiEddi-XML 1.3

- GUI-based tool for interactively editing feature models
- Feature models and feature diagrams are part of a repository
  - The repository is stored in XML format
- Each feature model is associated with a meta-model for additional information
  - The meta-model can be edited
  - This way it is possible to capture additional information
  - Some information can also be visualized in a feature diagram

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# AmiEddi-XML 1.3

- Developers
  - Mario Selbig, feature model editor
  - Frank Blinn, configurable meta-model, *www.informatik.fh-kl.de/~eisenecker/studentwork/dt_blinn.pdf*
  - Markus Lang, XML repository
- Development of XML repository was funded by Delta Software Technology GmbH, w*ww.d-s-t-g.com*
- Free download from *www.generative-programming.org*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Practice

- On demand
  - Demonstration of AmiEddi-XML 1.3

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Overview

- Elements and Notation
- AmiEddi-XML 1.3
- ➢ Captain Feature
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# New Requirements ...

... emerged from applying feature modeling and AmiEddi in real-world projects

- Modularization of complex feature diagrams
- Flexible cardinalities of features and groups
- Extending the meta-model by the user
- Specialization of feature diagrams and creating system configurations
- Assigning values to features during specialization

➔ Integrating feature modeling, meta-modeling, and specialization of feature diagrams as well as creating system configurations

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

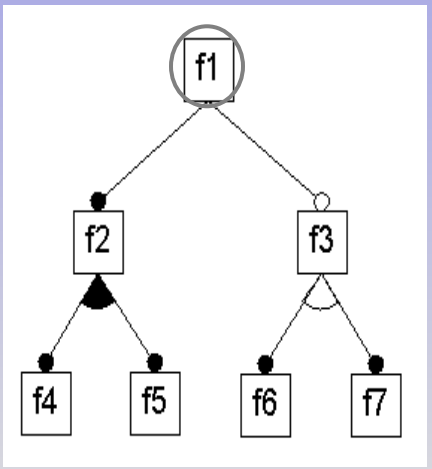


# Captain Feature

- Captain Feature is the successor of AmiEddi-XML and meets the afore-mentioned requirements
- Joint work with DaimlerChrysler Research Ulm
- Development
  - Thomas Bednasch: Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmodellierung, *www.informatik.fh-kl.de/~eisenecker/studentwork/dt_bednasch.pdf*
  - Markus Lang
  - Henrik Becker
  - James Markle
- Recently Captain Feature became an open source project
  - *https://sourceforge.net/projects/captainfeature/*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Elements of Feature Diagrams




- Tree structure
- Exactly one concept node
- Mandatory feature
- Optional feature
- Inclusive-or group
- Exclusive-or group
- New requirements
  - flexible groups
  - feature cardinalities

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Technical Concepts

- Feature
  - Comprises all properties of a feature
  - Is never used in diagrams and thus not visible in diagrams
  - Can be referenced in an arbitrary number of diagrams
- Feature node
  - The "visible" representation of a feature in a diagram
  - References a feature or - because of modularization - a concept node (representative)
  - During specialization a value can be assigned to it

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Technical Concepts

- Cardinality
  - Denotes, how often a representative may appear in a configuration
  - AmiEddi allows 0-1 and 1-1 (optional and mandatory feature)
  - Captain Feature has flexible cardinalities, e.g. 2-4
- Group cardinality
  - Denotes, how many variants of a group may appear in a configuration
  - AmiEddi allows exclusive-or and inclusive-or groups only
  - Captain Feature has flexible groups, e.g. 6 out of 49

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Concepts in EBNF

| | | |
|---|---|---|
| <Feature> | := | Name Description [Premature] [<Type>] |
| <Type> | := | (Bool \| Int \| Float \| String \|Pointer) |
| <FeatureNode> | := | Pointer [<Cardinality> \| <Group>]* |
| <Cardinality> | := | Minimum [Maximum] <FeatureNode> *) |
| <Group> | := | Minimum [Maximum] <Cardinality> <Cardinality>+ |

*) Expect some changes of the cardinality concept in near future

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Meta-Model as Feature Model

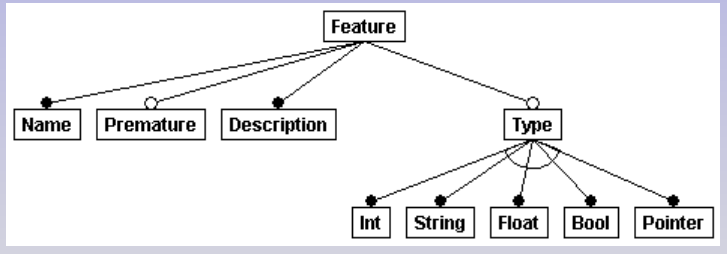
- Idea
  - Use feature diagrams to describe the technical concepts
  - These so called "system diagrams" are the core of the meta-model
- Consequence
  - The designer can use the feature model editor to enhance the meta-model
  - This turns feature modeling into a complete, self-descriptive system

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# System Diagram "Feature"

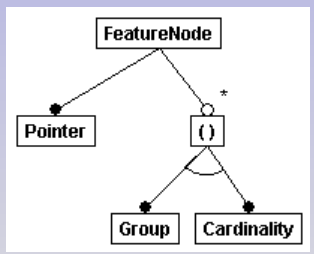
<Feature> := Name Description [Premature] [<Type>]




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# System Diagram "FeatureNode"

<FeatureNode> := Pointer [<Cardinality> | <Group>]*



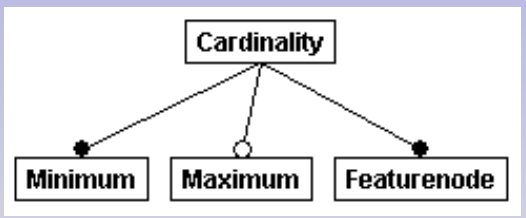

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# System Diagram "Cardinality"

<Cardinality> := Minimum [Maximum] <FeatureNode>

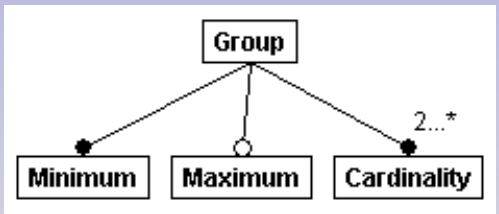
- Expect some changes of the cardinality concept in near future




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# System Diagram "Group"

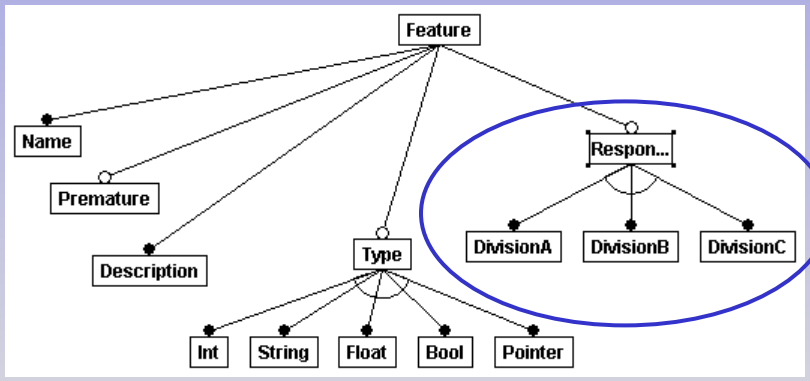
<Group> := Minimum [Maximum] <Cardinality> <Cardinality>+




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Extending the Meta-Model (Example)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

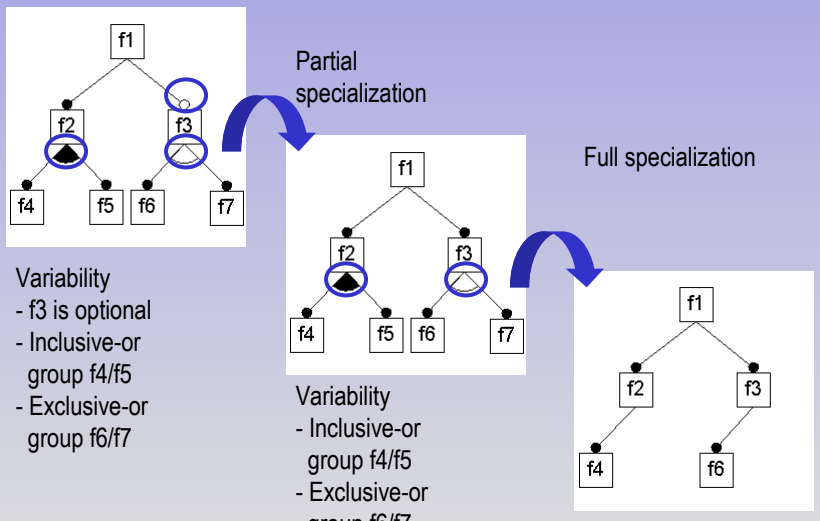
# Specialization and Configuration

- Specialization means to remove variability of a feature diagram
  - Breaking groups
  - Denoting the exact cardinality
- The (partial) specialization of a feature diagram produces a new feature diagram
- Any feature-diagram may be declared as configuration, even if it stills contains variability
  - Either the generator decides itself whether to resolve remaining variability during generation or to include the variability in the generated systems
  - or the binding time has to be provided explicitly as additional information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Specialization (Example)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Specializing the Meta-Model

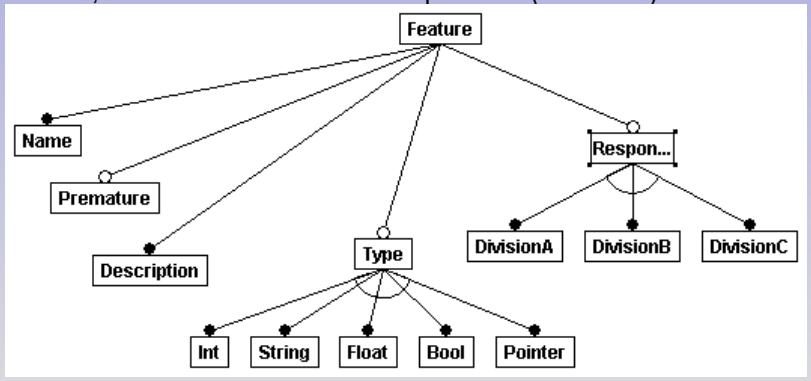
- The meta-model is also rendered as a feature model, thus specialization can be applied to the meta-model as well
- The specialization of system diagrams creates the elements a feature diagram consists of

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



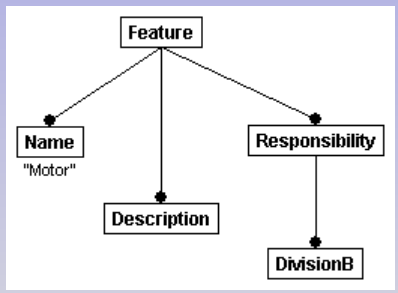
# Specializing the Meta-Model (Example)

Specializing this system diagram could produce a feature named “Motor", for which "DivisionB" ist responsible (next slide)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

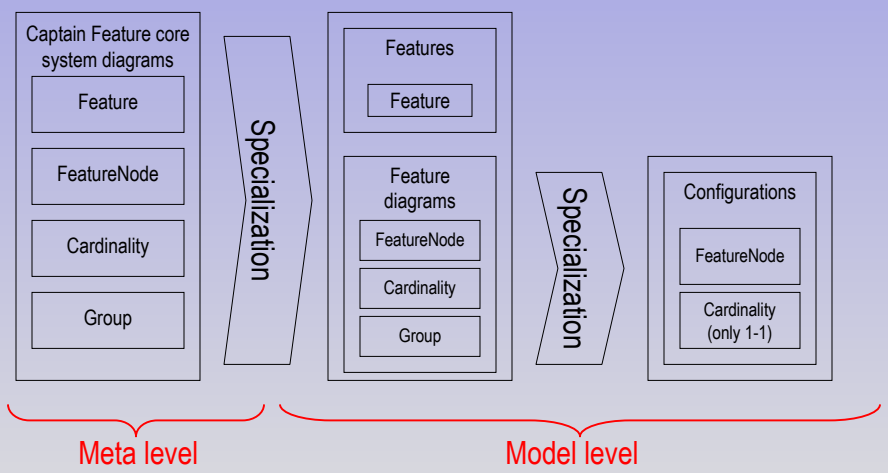
# Specializing the Meta-Model (Example)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Specialization (Overview)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Elements and Notation
- AmiEddi-XML 1.3
- Captain Feature
- ➢ Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# References


- Download of AmiEddi-XML 1.3 from *www.generative-programming.org*
- Mario Selbig: A Feature Diagram-Editor – Analysis, Design, and Implementation of its Core Functionality. Diploma thesis,Dept.of I/MST, FH Kaiserslautern, Zweibrücken 2000
- Frank Blinn: Entwurf und Implementierung eines Generators für Merkmalmetamodelle. Diploma thesis, Dept. of I/MST, FH Kaiserslautern, Zweibrücken 2001, *www.informatik.fh-kl.de/~eisenecker/studentwork/dt_blinn.pdf*
- Ulrich W. Eisenecker, Mario Selbig, Frank Blinn, Krzysztof Czarnecki: Merkmalmodellierung für Softwaresystemfamilien. In: OBJEKTspektrum 5/2001
- Thomas Bednasch: Konzept und Implementierung eines konfigurierbaren Metamodells für die Merkmalmodellierung. Diploma thesis, Dept. of I/MST, FH Kaiserslautern, Zweibrücken 2002, *www.informatik.fh-kl.de/~eisenecker/studentwork/dt_bednasch.pdf*


© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Practice

- On demand
  - Demonstration of "Captain Feature"
  - Feature modeling
  - Modularization of diagrams
  - Cardinalities of features and groups
  - Specializing features
  - Extending the meta-model

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Generative Programming

## Methods, Techniques, and Applications










Frames

# Overview

- Introduction
- Example with ANGIE
- Example with XFramer
- Technology Projection GP→Frames
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Background

- Marvin Minsky: A Framework for Representing Knowledge. MIT A.I. Laboratory, Artificial Intelligence Memo No. 306, June 1974, *ftp://publications.ai.mit.edu/ai-publications/0-499/AIM-306.ps*
  - Frames as flexible representation formalism for image processing and speech analysis
  - A frame defines common parts, which are present in each instance of a frame, and so-called "slots", which contain variable parts of a frame instance
  - Frames are used for pattern-based classification of scenes and texts
  - Frames bear interesting resemblance to feature modeling

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

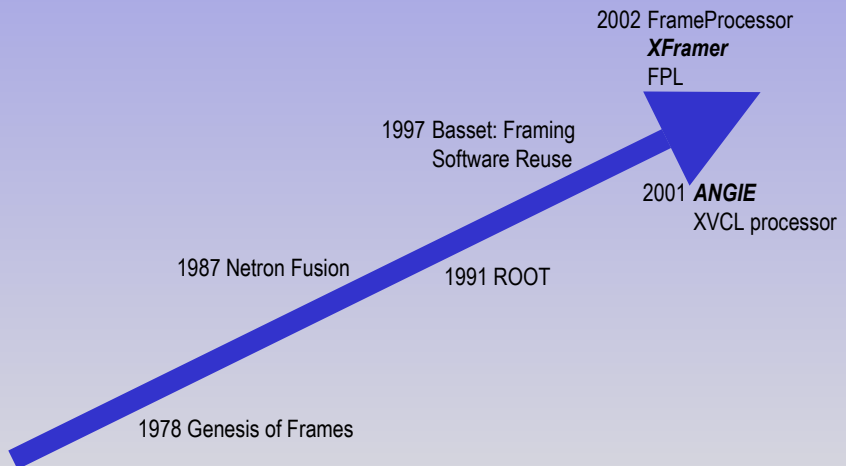
# Background

- Minsky's idea of frames attracted wide interest in the AI community and in cognitive psychology, but had no impact on software engineering
- Eventually 20 years later Basset described how to use frames for generating software
  - Paul G. Basset: Framing Software Reuse. Lessons From the Real World. Prentice-Hall 1997

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Historical Development




Eisenecker, Emrich, Endler, Schlee: Emerging Product Line Implementation Technologies: C++, Frames, and Generating Graphical User Interfaces, *www.polite-project.org*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Terms

- Note: The following statements are primarily about ANGIE and XFramer, some parts also apply to other frame processors
- Frame
  - Common text (host code) of all frame instances (optional)
  - Functions for creating and processing frame instances (in XFramer only)
- (Script-) Functions
  - Functions for creating and processing frame instances (ANGIE and XFramer)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Terms

- Frame instance
  - Exemplar of a frame
- Frame hierarchy
  - Hierarchical structure of frame instances (ANGIE, XFramer)
- Export
  - Production and output of source code

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

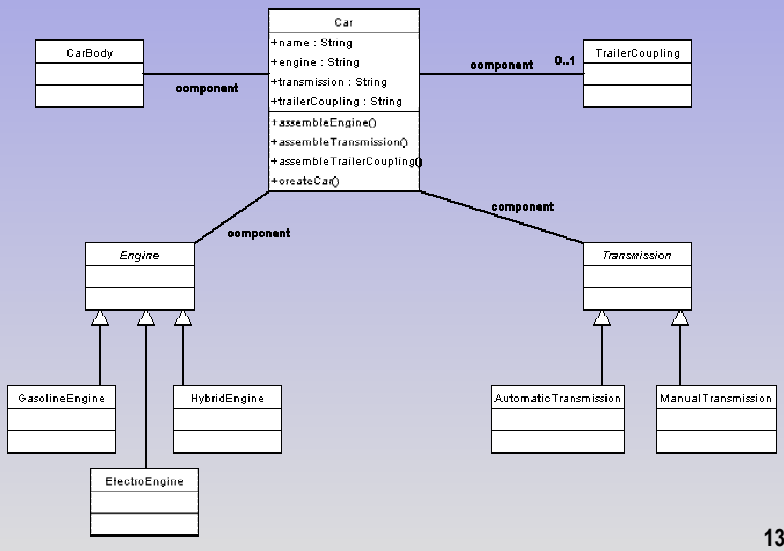
# Comparison of Classes and Frames

- Frames resemble classes
  - They are blueprints
  - Their instantiation creates objects and frame exemplars
  - Attributes correspond to slots
  - Methods correspond to (script-) functions
  - Frames can also contain additional test (host code) with embedded references to slots
  - There is no inheritance among frames
- Thus it is also feasible to use UML for modeling frames

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# UML-Class Diagram of Car Generator




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Overview

- Introduction
- ➢ Example with ANGIE
- Example with XFramer
- Technology Projection GP→Frames
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# ANGIE

- Product of Delta Software Technology GmbH
- Onomatopoeic for the acronym ***A N****ew* ***G****enerator* ***E****ngine*
- Second generation frame processor specialized for text generation
- Its predecessor ROOT can be applied for code analysis as well
- Download of freely available version *ANGIE Generation Now!* from *www.angie.d-s-t-g.com*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Implementation in ANGIE

- No inheritance between frames
  - All abstract classes are removed
  - All other classes are implemented as frames
- All attributes are turned into slots
- Association "Component" is implemented as collection
- No encapsulation of frames and methods
  - All methods are implemented as free (script-) functions
- For simplicity only a rudimentary DSL will be implemented
- No constructibility checking

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Implementation in ANGIE

```
.Frame GasolineEngine
°
struct GasolineEngine {
 GasolineEngine () {
    cout << " GasolineEngine ";
  }
};
.End Frame
```

**Legend**
- *Frame code*
- Host code
- ° Indentation flag (configurable), forces insertion of empty line

```
.Frame AutomaticTransmission
°
struct AutomaticTransmission {
 AutomaticTransmission () {
    cout << " AutomaticTransmission ";
  }
};
.End Frame
```

```
.Frame TrailerCoupling

struct TrailerCoupling {
 TrailerCoupling () {
    cout << " TrailerCoupling "
  }
};
.End Frame
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Implementation in ANGIE

```
.Frame Car(Name)
.Dim Components = createCollection()
.Dim Engine
.Dim Transmission
.Dim TrailerCoupling
#ifndef <!Name!>_H
#define <!Name!>_H
°
#include <iostream>
using namespace std;
°
struct CarBody {
 CarBody () {
     cout << " CarBody ";
  }
};
<!Components!>
°
```

```
struct <!Name!> {
   <!Name!> () {
     cout << "<!Name!>" << endl;
   }
   CarBody cb;
   <!Engine!> e;
   <!Transmission!> t;<?
   <!TrailerCoupling!> tc;?>
};
#endif
.End Frame
```

**Legend**
- *Slot Declaration*
- **Slot Reference**
- Optional Code block

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Implementation in ANGIE

```
.Function assembleEngine(Car,Engine)
  .Car.Engine = Engine
  .add(Car.Components,createFrame(Engine))
.End Function

.Function assembleTransmission(Car,Transmission)
  .Car.Transmission = Transmission
  .add(Car.Components,createFrame(Transmission))
.End Function

.Function assembleTrailerCoupling(Car)
  .Car.TrailerCoupling = "TrailerCoupling"
  .add(Car.Components,createFrame("TrailerCoupling"))
.End Function
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Implementation in ANGIE

```
.Function createCar(Name,Engine,Transmission,Options)
.Dim aCar = createFrame("Car",Name)

 .If Engine = "Gasoline" Then
   .assembleEngine(aCar,"GasolineEngine")
 .Else
   .If Engine = "Electro" Then
     .assembleEngine(aCar,"ElectroEngine")
   .Else
     .If Engine = "Hybrid" Then
       .assembleEngine(aCar,"HybridEngine")
     .Else
       .Print("Invalid engine specification: " & Engine)
       .End If
     .End If
 .End If
...
```

Engine assembly

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Implementation in ANGIE

```
...
 .If Transmission = "Manual" Then
   .assembelTransmission(aCar,"ManualTransmission")
 .Else
   .If Transmission = "Automatic" Then
     .assembleTransmission(aCar,"AutomaticTransmission")
   .Else
     .Print("Invalid transmission specification: " & Transmission)
   .End If
 .End If
 .If Options = "none" Then
 .Else
   .If Options = "TrailerCoupling" Then
     .assembleTrailerCoupling(aCar)
   .Else
     .Print("Invalid options specification: " & Options)
     .End If
 .End If
 .createCar = aCar
.End Function
```

Transmission assembly

Options assembly

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Implementation in ANGIE

```
.Function CAR_GENERATOR(Name,Engine,Transmission,Options)
  .export(createCar(Name,Engine,Transmission,Options),Name,"CPP Header")
.End Function
```

**Legend**

- *createCar* creates an instance of frame *Car* and assigns parameters to its slots
- **export** writes the frame instance in a source file; the source file has name *Name* and extension *.h* (according to the parameter "CPP Header")

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Practice

- On demand
  - Demonstration of the car generator implemented in ANGIE
  - Extending the car generator

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Overview

- Introduction
- Example with ANGIE
- ➢ Example with XFramer
- Technology Projection GP→Frames
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# XFramer

- Public domain software developed by Max Schlee
- Adds frame processing to C++, Java, and C#
  - One preprocessor for all languages
  - Language specific libraries
  - Executing *xframer framefile.lang generator.lang* preprocesses frame commands of *framefile.lang* and creates *generator.lang*
  - After compiling *generator.lang* it can be executed to generate the desired text or binary code
- Download from *www.geocities.com/mslerm/xframer*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Implementation in XFramer

- No inheritance between frames
  - All abstract classes are removed
  - All other classes are implemented as frames
- All attributes are turned into slots; any variable of each elementary type or type *MString* can be used as slot
- Association "Component" is implemented as STL collection
- Currently, optional code blocks don't exist
- A frame encapsulates slots, host code, and methods
- For simplicity only a rudimentary DSL will be implemented
- No constructibility checking

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Implementation in XFramer

```
frame (GasolineEngine) {
  public:
    GasolineEngine() {
BEGIN_FRAME_TEXT()
struct GasolineEngine {
 GasolineEngine () {
    cout << " GasolineEngine ";
  }
};
END_FRAME_TEXT()
    }
};
```

**Legend**
- *Frame code*
- Host code

```
frame (AutomaticTransmission) {
  public:
 AutomaticTransmission() {
BEGIN_FRAME_TEXT()
struct AutomaticTransmission {
 AutomaticTransmission () {
    cout << " AutomaticTransmission ";
  }
};
END_FRAME_TEXT()
    }
};

frame (TrailerCoupling) {
// ...
```

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Implementation in XFramer

```
frame (Car) {
  public:
    Car(char* name,const string& engine, const string& transmission,const string& options) {
      if (engine == "Gasoline") {
        Engine = "GasolineEngine";
        Components.push_back(GasolineEngine());
      }
      else if (engine == "Electro") {
            Engine = "ElectroEngine";
            Components.push_back( ElectroEngine());
      }
      else if (engine == "Hybrid") {
            Engine = "HybridEngine";
            Components.push_back( HybridEngine());
      }
      else cerr << "Invalid engine specification: " << engine.c_str() << endl;
            Components.push_back("\n\n");
```

Engine assembly

**Legend**
- *Slot- declaration*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Implementation in XFramer

```
      if (transmission == "Manuak") {
        Transmission = "ManualTransmission";
        Components.push_back(ManualTransmission());
      }
      else if (transmission == "Automatic") {
            Transmission = "AutomaticTransmission";
            Components.push_back(AutomaticTransmission());
      }
      else cerr << "Invalid transmission specification: " << transmission.c_str() << endl;

      if (options == "none") {
      }
      else if (optiones == "TrailerCoupling") {
            TrailerCoupling = "\n TrailerCoupling tc;";
            Components.push_back("\n\n");
            Components.push_back(TrailerCoupling());
      }
      else cerr << "Invalid option specification: " << options.c_str() << endl;
```

Transmission assembly

Options assembly

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Implementation in XFramer

```
BEGIN_FRAME_TEXT()
#ifndef <!name!>_h
#define <!name!>_h

#include <iostream>
using namespace std;

struct CarBody {
 CarBody () {
    cout << " CarBody ";
  }
};
<!toMString(Components)!>
```

```
struct <!name!> {
  <!name!>() {
    cout << "<!name!>" << endl;
  }
  <!Engine!> e;
  <!Transmission!> t;<!TrailerCoupling!>
}
#endif <!name!>_h
END_FRAME_TEXT()
    }
  private:
    MString Engine,Transmission,
            TrailerCoupling;
    vector<MString> Components;
};
```

**Legend**
- *Slot declaration*
- **Slot reference**

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Implementation in XFramer

```
int main(int argc,char* argv[],char* envp[]) {
  if (argc != 5) {
    cout << "Invalid number of arguments." << endl
         << "Use car Name Engine Transmission Options" << endl;
    return 1;
  }
  string filename = "c:\\ecoop\\"; filename = filename + argv[1] + ".h";
  Car(argv[1],argv[2],argv[3],argv[4]).exportToFile(filename.c_str());
  cout << "Export complete" << endl;
  return 0;
}
```

**Legend**
- *Car()* creates an instance of frame Car and assigns parameters to slots
- **exportToFile** writes the frame instance as textfile with name *filename.c_str()*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Practice

- On demand
  - Demonstration of the car generator implemented in XFramer
  - Extending the car generator

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Overview

- Introduction
- Example with ANGIE
- Example with XFramer
- ➢ Technology Projection GP→Frames
- Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Frames and DSLs

- The previous examples don't apply special DSLs
- Embedded text-based DSLs can be implemented using frames and functions only
- Mostly an analyzed specification will be passed as parameter to a frame processor
  - Text file containing keyword parameters
  - Text file containing position-based parameters
  - XML file
- Usually a separate program, e.g. a GUI-based dialog, is applied to create such a specification

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Frame-Based Architectures

- The architecture of the previous examples is aligned to that of the car generator implemented in C++
  - Implementation components are strictly separated from configuration generator
  - Larger configurations result in a complex code, which is difficult to maintain
- Frame instances usually conform to a tree or a directed acyclic graph
  - They exhibit a structure with a striking resemblance to feature diagrams

  ➔Idea: adopt the structure of the feature diagram as frame-based architecture!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

## Feature Diagram-Based Specification

- Comprises either
  - an entire feature diagram, whereby selected features are weighed with 1 (or true) and rejected features with 0 (or false), or a
  - Configuration of a feature model containing no more variability
- In both cases an XML-based solution is well-suited

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

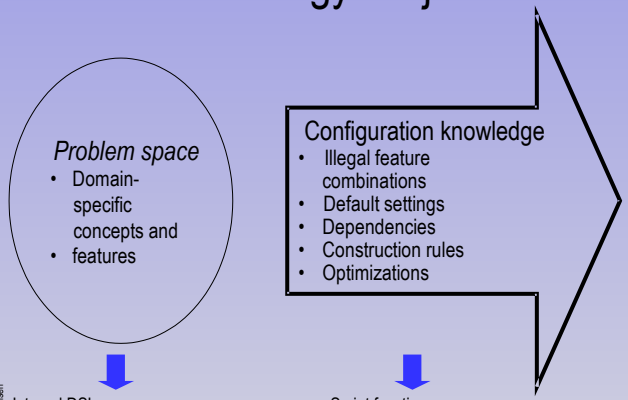


## "Intelligent Frames"

- The appropriate (sub-) tree is passed as argument to a frame instance
- The frame instance knows the relevant configuration information to parameterize its slots
- If required, new frame instances will be created, and the appropriate (sub-) tree will be passed to them
- In contrast to the previously demonstrated localization in a single generator the configuration knowledge is now distributed among many frame instances

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Technology Projection GP→Frames




Internal DSLs
- Script functions
- Frames

External specifications
- Text files
  - Keyword (named) parameters
  - Position parameters
- XML

- Script functions
- Frames (only to a lesser extent)

Components
- Frames
- Script functions (only to a lesser extent)

- Different from C++ a GenVoca architecture is no longer required for implementation
- Interesting experience with tree/DAG like architectures that parallelize the feature graph

Architecture
- Frames parameterized from outside
- Self-configuring ("intelligent") frames

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Comparison C++ and Frames

**C++**

- C++ is a two-level language
- Template-metaprograms are part of the language
- Seamless integration of generative library and client code
- Only C++-code can be generated

**Frames**

- Frame processing requires separate execution of frame processor
- Frames and functions do not know about the processed host code (with some exceptions of XFramer)
- Frames can be used to generate source code, GUI resources, help texts, documents, etc.

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# ABA and MiniABA

- ABA, generating image viewers
  - Technical code
  - GUI code
  - Help text
  - Make files
  - Platform and tools: Windows, VC++ 6.0, Qt 2.3.0, ANGIE R2.1
- MiniABA, generating calculators
  - Technical code
  - GUI resources
  - Make files
  - Platform and tools: Windows, VC++ 6.0, CFramer
- Max Schlee: Generative Programming of Graphical User Interfaces. Diploma thesis, Dept. of I/MST, FH Kaiserslautern, Zweibrücken 2002, *www.informatik.fh-kl.de/~eisenecker/studentwork/dt_schlee.pdf*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Overview

- Introduction
- Example with ANGIE
- Example with XFramer
- Technology Projection GP→Frames
- ➢ Further Information

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# References


- Basset: Framing Software Reuse: Lessons From the Real World. Prentice-Hall, 1996
- Eisenecker/Schilling: Generative Programmierung mit einem Frameprozessor. In iX 10/2002, *www.delta-software-technology.com/neu/media/pdf/facts_d/dlt11474.pdf*
- Emrich, Schlee, Eisenecker: PoLITe-Survey, *www.polite-project.org*
- Sauer: Metadata driven multi-artifact code generation using Frame Oriented Programming, *www.softmetaware.com/oopsla2002/sauerf.pdf*


© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Download

- ANGIE Generation Now!, *www.angie.d-s-t-g.com*
- XFramer, *www.geocities.com/mslerm/xframer*
- FPL, *http://sourceforge.net/projects/fpl*
- FrameProcessor, *http://sourceforge.net/projects/frameprocessor*
- Netron Fusion, *www.netron.com/products/fusion* (product information only, no trial or public domain version)
- XVCL processor, *http://sourceforge.net/projects/fxvcl/*

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Practice

- On demand
  - Demonstration of ABA
  - Demonstration of MiniABA

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Generative Programming

## Methods, Techniques, and Applications










Model-Driven Architecture

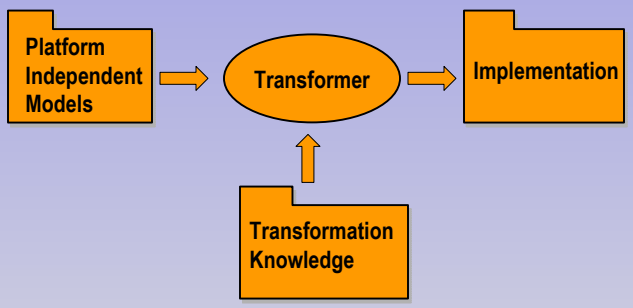
# Outline

- MDA Basics
- Metamodeling
- Model Transformation
- Example
- Discussion and Further Readings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



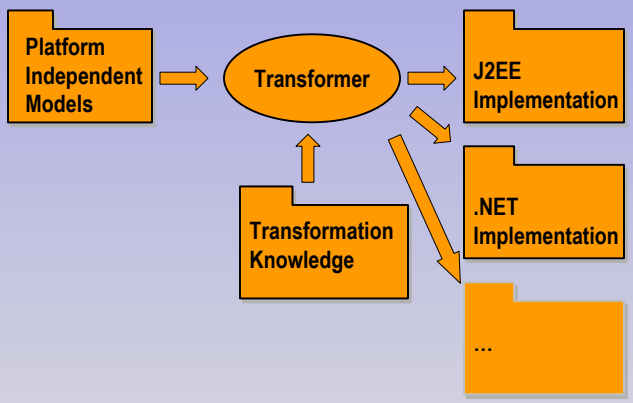
# MDA From 30.000 Feet




- Use of *platform independent models* (PIM) as specification
- Transformation into *platform specific models* (PSM) using tools

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# MDA From 30.000 Feet

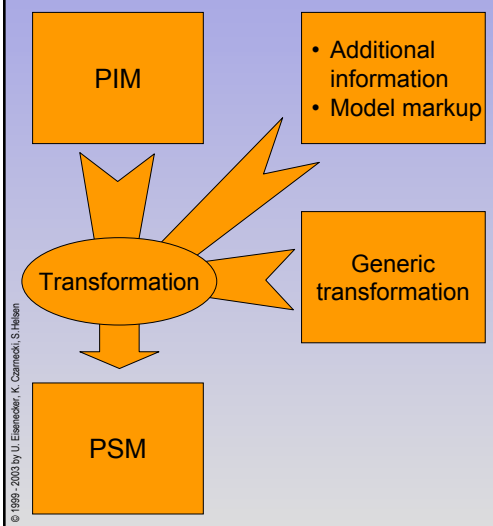



- A PIM can be retargeted to different platforms
- Not the only reason why MDA might be of interest to you…

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



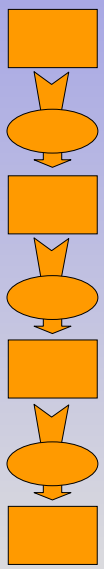
# Basic MDA Pattern




- Generic transformations
  - Implement best practices, architectural and design patterns, technology patterns (e.g., J2EE patterns), optimizations, etc.
- Additional information
  - Adjust the transformation globally
  - Similar to compiler options
- Model markup
  - Direct the transformation of particular model elements
  - Not part of the PIM
  - Different platform mappings may require different markup
  - Similar to compiler pragmas

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Basic MDA Pattern



- The basic pattern can be applied multiple times
- PIMs and PSMs are relative notions
  - "Someone's PIM can be someone else's PSM"
- Platform independence is relative, too
  - It's a scoping issue
  - It's a strategic decision

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

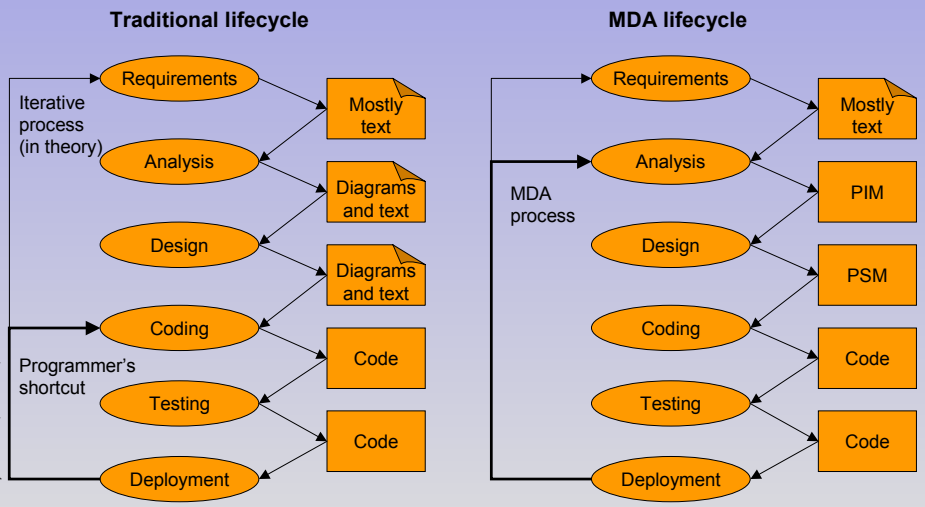


# Role of Models

- Capture design information that is usually absent from code and lost during development
- Basis for
  - System generation
  - Analysis
  - Simulation
  - Test generation
  - Documentation generation
  - …
- *Domain-specificity* of a modeling language strengthens its capabilities for generation, optimization, early error detection, etc.

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Impact of MDA on the Development Process




Source: Kleppe et al 2003

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# MDA and Agile Development

- MDA is appropriate for agile development
- Models are not just additional documentation artifacts, but they are the actual source
- Instant feedback through simulation / rapid code generation
- Model-based testing
- Domain-specific modeling languages may simplify communication with your customer

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Separation of Concerns in MDA

- PIM development
- Mapping decisions
  - Markup by an architect
- Development of DSLs and reusable transformations
- Platform development
- Development of modeling tools and generator infrastructures

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# MDA-Related Standards

- OMG Standards
  - Modeling – UML
  - Metamodeling – MOF
  - Action semantics
  - Model interchange – XMI
  - Diagram interchange
  - Human-readable textual notation – HUTN
  - Model-based testing and debugging
  - (CWM)
  - …
- Java Community Process (JCP) Standard
  - Java Metadata Interface – JMI

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% traceability from specification to implementation

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Outline

- MDA Basics
- ➡ Metamodeling
- Model Transformation
- Case Study
- Tools
- Discussion and Further Readings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

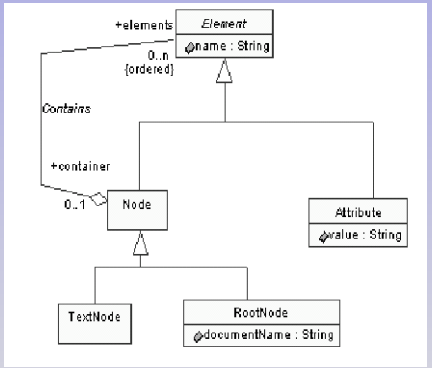
# Metamodeling

- Metamodel
  - Model of a modeling language
  - Definition of syntax and semantics
- Meta Object Facility (MOF) provides a set of concepts to define metamodels; in particular
  - Class diagrams to define abstract syntax and
  - OCL to define semantics of a modeling language
- Example: UML Metamodel
  - Semantics is defined using a mixture of OCL and informal text
- Example: Metamodel of XML

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# MOF Metamodel of XML




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Relationship Between MOF and UML

- MOF is distinct from UML, but for most practical purposes it can be viewed as a subset of the UML class model notation
- UML class modeling constructs missing in MOF include
  - Association classes
  - Quantifiers
  - Dependencies
  - N-ary associations (will become available in MOF 2.0)
- Alignment between MOF and UML
  - MOF 2.0 metamodel is a true subset of UML 2.0 metamodel
  - Not the case in previous versions

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Standard Technology Mappings for MOF

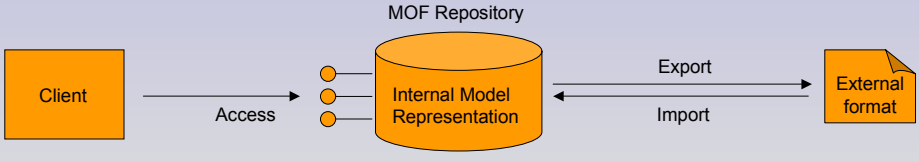
**MOF Metamodel**

**APIs for model manipulation**

- Java Metadata Interface – JMI (Java mapping)
- CORBA IDL mapping (see the MOF spec)
- WSDL mapping (in progress)
- …

**Serialization of a model**

- XML Metadata Interchange – XMI (XML mapping)
- Human Usable Textual Notation – HUTN
- …




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# The Role of UML in MDA

- MDA does not require UML
  - New modeling languages can be defined using MOF
- Applications of UML in MDA
  - General-purpose modeling language
  - Basis for extension and reuse
  - A way to provide concrete graphical syntax with tool support today (will loose importance in the long run)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Approaches to Defining Modeling Languages in MDA

- Lightweight UML extension
  - Extend UML through a profile
  - Appropriate for languages that are very close to UML
  - Extension stays within the UML semantics
- Heavyweight UML extension
  - Extend the UML metamodel directly through MOF mechanisms
    - E.g., by defining new subclasses in the metamodel)
  - Most appropriate if a significant extension necessary and/or only some parts of a metamodel need to be reused
- Create a new MOF metamodel
  - Appropriate for languages that are completely different from UML
    - May still reuse some parts of UML model per copy&paste

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Outline

- Motivation and MDA Basics
- Metamodeling
- ➡ Model Transformation
- Case Study
- Tools
- Discussion and Further Readings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Role of Model Transformations in MDA

- Model compilation
  - Automatic PIM to PSM
- Model query and view
  - Synchronization between models (propagation of change)
    - Different levels of abstraction (high-level vs. detailed)
    - Different system aspects (e.g., business objects, workflow)
- Model evolution
  - PIM to PIM (e.g., refactoring)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Approaches to Model Transformations

- Major categories of approaches [Czarnecki&Helsen03]
  - Model-to-code
    - Visitor-based
    - Template-based (most MDA tools today)
  - Model-to-model
    - Direct manipulation (e.g., through JMI)
    - Relational approach (aka logic-based programming)
    - Graph-transformation approaches
    - Structure-driven approaches (source or target)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Approaches to Model Transformations

- Several areas of variation, e.g.,
  - Representation of transformation rules
    - Declarative and/or imperative logic; use of patterns (graph or string)
  - Application control (where in model)
  - Scheduling mechanisms (execution order of transformations)
  - Reversibility (esp. for synchronization)
  - Reuse and extension mechanisms for transformations
  - Modularity

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Discussion

- OMG is working on a standard for defining transformations, known as Query/View/Transformation (QVT)
- Most MDA tools provide model-to-code transformations based on code templates (aka JSP)
- Some tools provide a framework for model-to-model transformations
- Graph-based approaches are in research stage
- No satisfactory solutions for synchronization yet

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Outline

- MDA Basics
- Metamodeling
- Model Transformation
- ➡ Example
- Discussion and Further Readings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Example - Overview

- Transformation between UML class diagrams
  - Relatively simple and not very realistic
  - But…
    - Illustrates main ideas of MDA
    - Details of transformation already elaborate
- In the real world: EJB technology mappings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



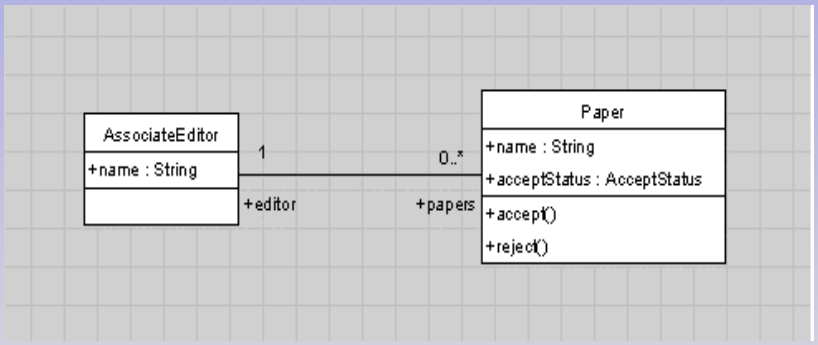
# Simple MDA Mapping

- *PIM*: UML class diagram
  - Analysis level
  - Classes, attributes, associations, operations
- *PSM*: UML class diagram
  - Implementation level
  - No associations, no public attributes

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# PIM Instance

Example class diagram at the analysis level




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



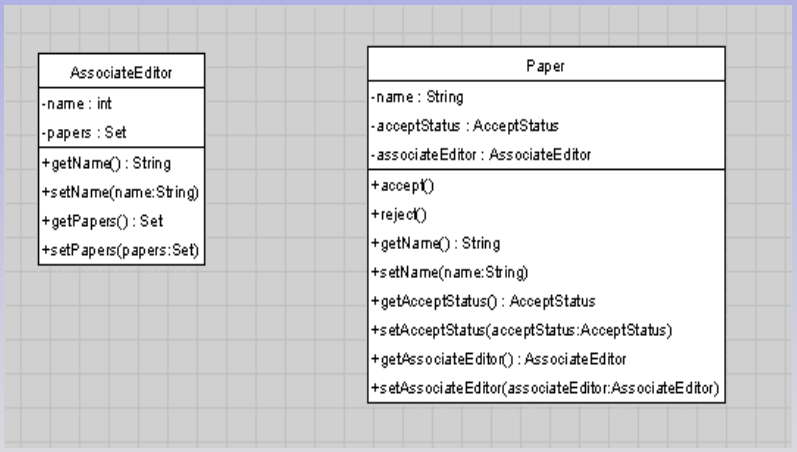
# Requested Changes

- Transformations
  - Public attributes become private
    - Access methods to private attributes
  - Association ends modeled as private attributes
    - Access methods for these new attributes
- Issues to address
  - What if multiplicity ≠ 1?
  - Form of access methods?

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# PSM Instance

Example class diagram at the implementation level




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



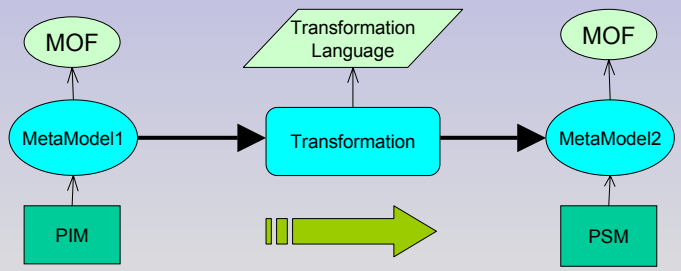
# PSM Decisions

- Decisions for PSM are domain and/or platform specific
- E.g., use `java.utils.HashSet` instead of `Set`
- MDA particularly strong whenever
  - Platform specific knowledge complex
  - Platform specific details mostly orthogonal to platform independent details

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Specifying Transformations

- How do we specify such a transformation?
- Recall the two main ingredients
  - PIM and PSM are modeled in a meta-modeling language (MOF-based)
  - Transformation between meta-models, written in a transformation language (e.g. MOF-QVT)




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

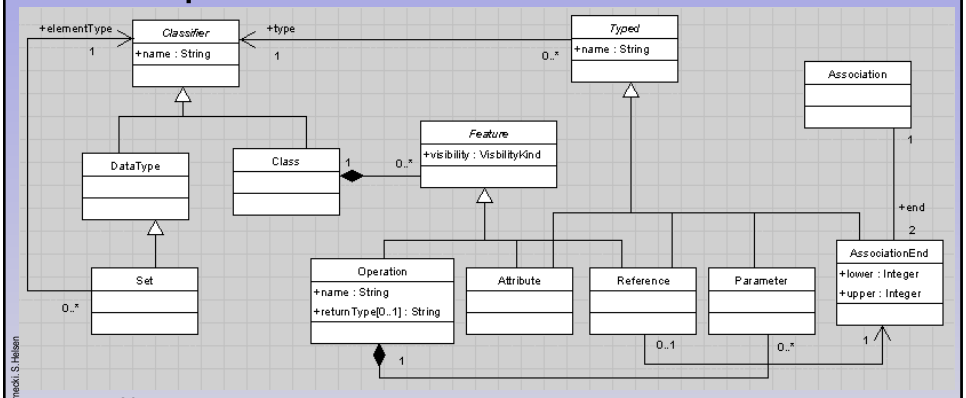


# For Our Example …

- … we provide
  - a simplified UML meta-model in MOF
  - a description of transformations in natural language
  - a more formal description based on graph transformations

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Simplified MOF Meta-model of UML




- Notes:
  - In the actual UML meta-model
    - many more classes and associations
    - many OCL constraints to refine semantics
  - PIM/PSM UML-diagrams: pretty-printed versions of instances of this meta-model.

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Transformation in Natural Language

- For each class in PIM, we have a class in PSM with the same name
- For each public attribute `attrName` in a PIM-class, we have the following features in the associated PIM-class:
  - A private attribute with name `attrName`
  - A public operation with name get`attrName`, no parameters, and the type of the PIM-attribute as return type
  - A public operation with name set`attrName`, no return type, and one parameter of the name `attrName` with a type equal to the type of the PIM-attribute.
- Etc.

⇒ Natural language is obviously non-executable …

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Graph Transformation Language

- For the purpose of our example, we select a simple graph transformation language (GTL)
  - Simplified graph transformation sufficient
  - More complex formalism may be required for more complex examples
    - E.g., the MOF-QVT proposals are fully text-based!
  - Informal Presentation
    - For a formal treatment, see a.o. [VarroVarroPataricza2002, AgrawalKarsaiShi2003, BraunMarschall2003, AppukuttanClarkReddyTrattVenkatesh2003, etc.]

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



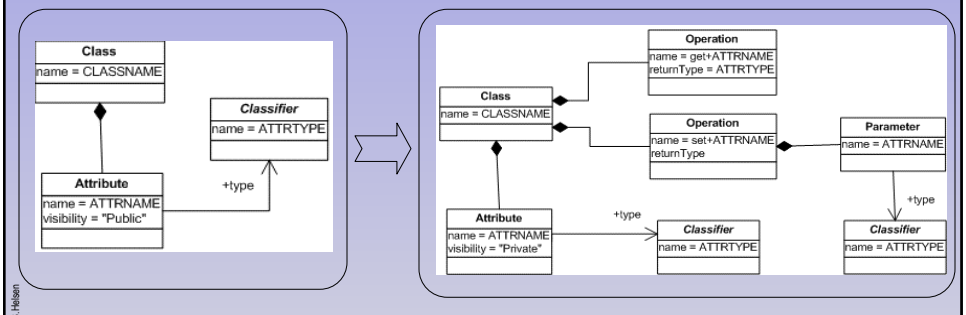
# GTL Informal Explanation




- Both LHS and RHS ≈ model graph patterns with (meta-) variables
- LHS pattern has possibly extra constraints
- RHS graph-pattern enriched with simple calculus on primitive types (e.g., string concatenation)
- All rules apply concurrently on all possible matches in PIM
- Model destructively transformed in place as follows:
  1. Keep classes/associations that match
  2. Remove classes and associations that do not match
  3. Add new associations and classes

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

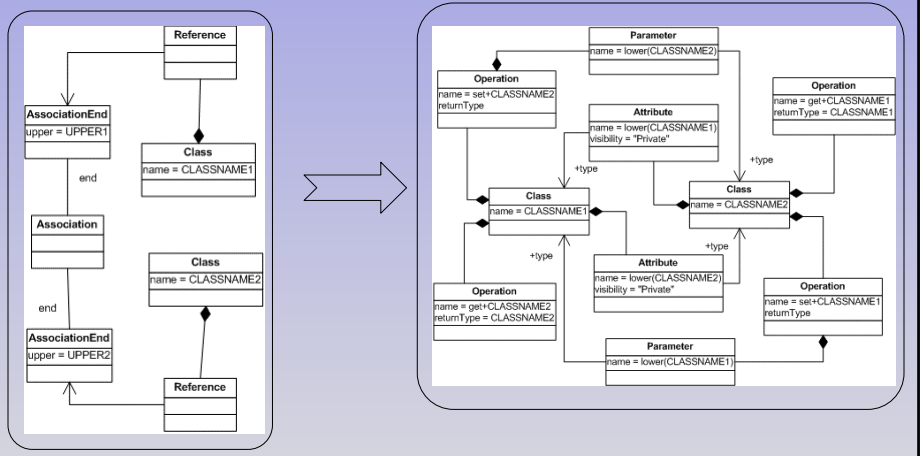
# Transformation Rule 1




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Transformation Rule 2




*UPPER1* ≤ 1 and *UPPER2* ≤ 1

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Remaining Rules

- The 3 other variations of Rule 2 where upper-bound > 1 are very similar
  - Attribute gets type `SET` instead
- Alternatively only two versions of Rule 2
  - However: requires more complicated rule scheduling and attribute assignment
- A lot of design space for a transformation language!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

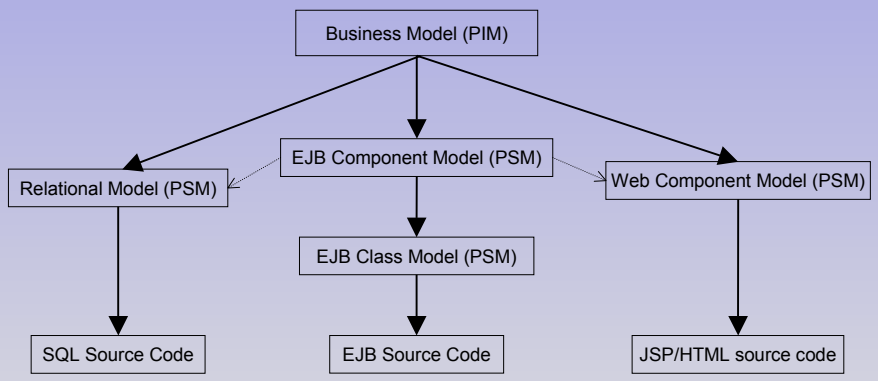


# Does This Really Scale?

- MDA in infancy and tools still evolving
- Graph transformations only one possibility
  - MOF/QVT proposals more concise
- A lot of focus on middleware mappings
  - e.g., EJBs, MS.Net, CORBA, etc.
- Many proposals geared towards EJB application development

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# A Typical EJB Application




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Typical EJB Application

- Transformations add platform expert knowledge to business model
  - E.g., the EJB component mapping contains EJB specific decisions
    - Grouping of classes to minimize networking
    - Assignment of primary keys
    - The use of container-managed beans
  - Generally, use well-known mappings and best practices (e.g. EJB patterns)
  - Tune transformations for intended target context

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# EJBs - Other Issues

- Static versus dynamic modeling of business model
- Incompleteness of generated PSMs and/or code
- Use of DSLs to describe and modularize PIM
- Complexity of describing actual transformations (e.g., what transformation language?)
- Commercially viable (e.g., OptimalJ, Codagen Architect, ArcStyler, etc.)

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# MDA in OptimalJ

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Outline

- MDA Basics
- Metamodeling
- Model Transformation
- Example
- ➡ Discussion and Further Readings

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

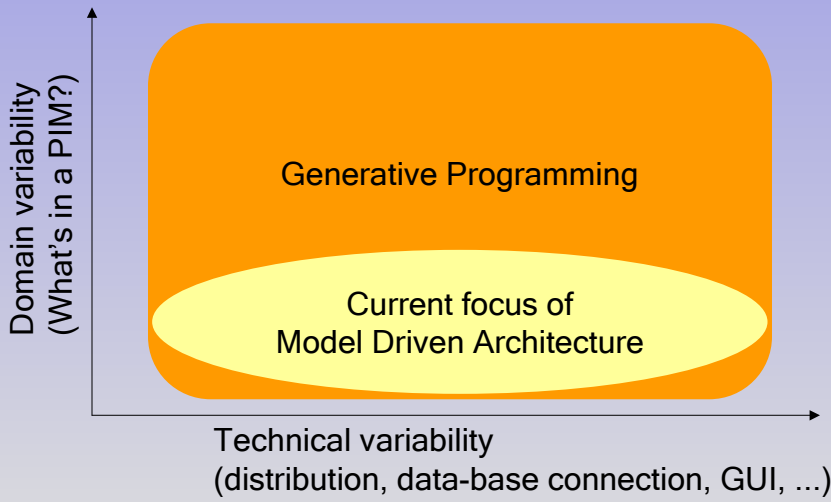


# GP and MDA

- **Generative programming** aims at modeling and implementing system families in such a way that a given system can be automatically generated from a specification written in a domain-specific language.
- Potential contribution to MDA
  - Systematic domain scoping and DSL development
    - Addresses important MDA questions
    - What is the right language for a PIM? What is a platform?
  - Advances in metaprogramming

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Relationship GP and MDA




© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Potential Benefits of MDA

- Preserving the investment in knowledge
  - Independent of implementation platform
  - Tacit knowledge made explicit
- Speed of development
  - Most of the implementation is generated
- Quality of implementation
  - Experts provide transformation templates
- Maintenance and documentation
  - Design and analysis models are not abandoned after writing
  - 100% traceability from specification to implementation

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Caveats

- The MDA vision is in its fast early evolution phase
  - Many important standards are still at an early definition stage (e.g., Query / View / Transformation)
  - Other standards (like UML 2.0 and MOF 02.) underwent significant evolution and tools need to catch up
- Existing tools are far still from realizing the MDA vision
  - Poor or no support for metamodeling, model transformation, and synchronization

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen



# Caveats

- Very few domain-specific modeling languages (or profiles) and platform mappings are implemented and available for reuse
  - Most of today's MDA tools target generating J2EE apps
- Many developers still prefer coding over working with modeling tools
  - The handling and efficiency of modeling tools is still far from that of programming IDEs

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S.Helsen

# Today

- Many of the available technologies can be put into good use
  - Don't write CRUD (create, read, update, delete) functionality by hand
  - A lot of infrastructure can be generated today
  - MOF and XMI provide metadata interoperability

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Future

- Merging of modeling and programming IDEs
  - The distinction between modeling and programming will be blurred
- Greater focus for domain-specific languages
  - Many DSLs and platform mappings will be available to choose from
  - DSLs will enable domain experts who are not programmers to build software
  - Programmers will focus on providing infrastructures and transformations
- Emergence of "software supply chains" (Jack Greenfield)
  - Greater specialization and reuse in the software industry
- MDA will not solve all problems
  - Interoperability
    - Will not be provided by MDA
    - Specific industries may develop their own standards in the long run

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Further Readings – Books

- Frankel. “Model Driven Architecture: Applying MDA to Enterprise Computing.” Wiley, 2003
- Kleppe, Warmer, & Bast. MDA Explained: The Model Driven Architecture--Practice and Promise. Addison-Wesley, 2003
- Hubert. “Convergent Architecture: Building Model Driven J2EE Systems with UML.” Wiley 2001
- Grose, Doney, & Brodsky. Mastering XMI: Java Programming with XMI, XML, and UML. Wiley, 2002
- Czarnecki & Eisenecker, “Generative Programming: Methods, Tools, and Applications.” Addison-Wesley, 2000
- Greenfield, “Model-Driven Development: Automating Component Design, Implementation, and Assembly.” Wiley, upcoming

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen



# Further Readings – Online

- MDA Guide
  - www.omg.com/mda
- DSTC MOF Pages
  - http://www.dstc.edu.au/Research/Projects/MOF/
- Online collection of metamodels
  - http://mdr.netbeans.org/metamodels.html
- Tool website
  - www.codegeneration.org
- Open source MOF repositories
  - http://mdr.netbeans.org
  - http://nsuml.sourceforge.net/

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Questions?

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

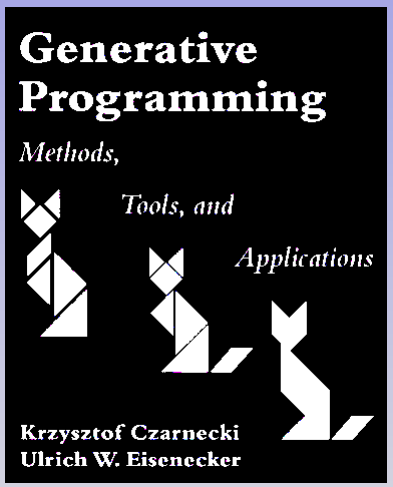


# Discussion

- Your questions, ideas and suggestions?
- Relation between AOP and GP
- Introducing system family development
- Identifying promising projects for GP
- Organizational issues
- Projects and experience with GP in industry
- ...

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen

# Thank You For Your Attention ...




… And Watch Out For These Cats!

© 1999 - 2003 by U. Eisenecker, K. Czarnecki, S. Helsen