

Product Lines and Variant Management



www.mdsd-buch.de



www.mdsd-book.org

Markus Völter

voelter@acm.org

www.voelter.de



Ingenieurbüro für Softwaretechnologie

www.voelter.de

- 1 -

© 2003 - 2006 Markus Völter

Software System Families

- Typically, MDD makes most sense in the context of **software system families** because developing modeling environments, generators, translators, etc. can be a lot of work and it pays only if reused.

- What is a **software system family**?

We consider a **set of programs** to constitute a family whenever it is worthwhile to study programs from the set by **first studying the common properties** of the set and **then determining the special properties of the individual family members**.

Definition by Parnas



Ingenieurbüro für Softwaretechnologie

www.voelter.de

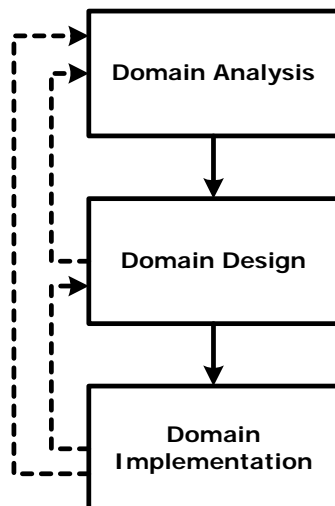
- 2 -

© 2003 - 2006 Markus Völter

Examples for Software System Families

- A set of **projects in the same domain** (banking, telecom switching, automotive diagnosis).
 - You might be able to generate recurring business logic from models.
- A set of **artifacts based on the same infrastructure** (such as EJB) in one project.
 - Here, you might be able to generate all the infrastructure-specific code around manually implemented business logic.
- you have some **specific business logic that you want to run on different platforms**.
 - You might be able to generate platform-specific implementation code from the models (this is the focus of MDA)
- a set of **artifacts based on the same modelling paradigm**, such as state chart.
 - You might be able to generate the complete implementation based on the model and its predefined mapping to lower-level implementations.

Product Line Engineering



- Domain Scoping
 - Variability Analysis
 - Domain Structuring
- Define common architecture
 - Define Production Plan
 - Define Building Blocks
- Components
 - DSLs & Generators
 - Production Process

Domain Analysis

- **Goal:** A domain model (aka metamodel)
- **Scoping** defines, **what is part of the domain, and what is not** (defines the range of possible products of the family)
- A **common vocabulary** defines the terms in which the domain can be described
- A result of this process is also the knowledge of **whether the domain is mature, or not** („are there more commonalities than differences?“)
- The **commonalities and the differences** between different products in the domain have to be defined
 - ➔ Variability Analysis



Variability Analysis

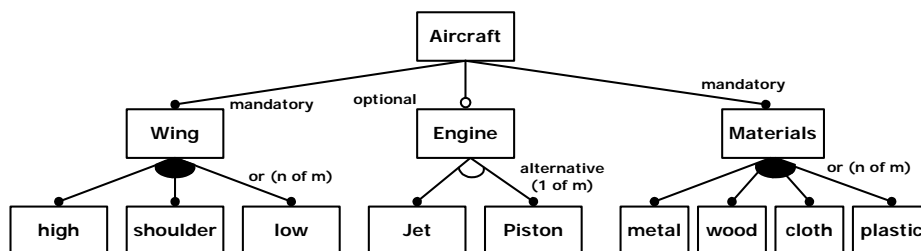
- **Variability analysis** discovers the variable and fixed parts of a product in a domain.
Parts can be
 - Structural or behavioral
 - Functional or non-functional (technical)
 - Modularized or aspectual
- To define variable parts, we need to **have a commonality base**: a base platform, a common architecture
- There are **two kinds of variability**:
 - positive variability: add something (optional)
 - negative variability: removes something (essential)
- Positive variability leaves the concept intact, while netative variability does not.



Feature Modeling

- As a consequence of the domain analysis and before defining a concrete metamodel, usually, a **feature modelling** phase makes sense to systematically define optional and mandatory features.
- A feature model **describes the (sub-)features of a concept**, subfeatures can be
 - Mandatory
 - Optional
 - Alternative
 - N of M
- A feature can represent some kind of **component** or an **aspect**.
- A feature model can be **multi-dimensional**
- A graphical notation exists: **feature diagrams**

Example Feature Diagram



- Example products:
 - An aircraft with a low wing, piston engine and made of metal, wood and cloth: **Robin DR-400**
 - An aircraft with shoulder wing, no engine and made of plastic: **ASW-27**
 - An aircraft with low wing, jet engine(s) and made of metal: **Airbus A320**



Example Feature Diagram II



Feature Diagrams cont'd

- They can contain **constraints** on the combinations of features
- They can define „**names**“ for specific combinations of features; feature groups
- Features can be **open**: additional subfeatures can be added
- Features can be **incomplete**: the subfeatures are not yet defined
- **Multiplicity of subfeatures** can be added
- And more...

Binding Time Analysis

- A feature diagram defines the common and variable parts of a system. It has to be determined **when it needs to be decided** if a specific (sub-) feature will be present in a specific product in the family. This is called **binding time**.
- Binding time has **consequences** on
 - flexibility
 - performance
 - code size
 - type safety
 - and: on the **technique used to implement** the variable aspect

Typical Binding Times

- **source time:** manual programming, template parameters, generators
- **Compile time:** function overloading, precompiler, template evaluation
- **deployment/configuration time:** component deployment (impl. for an interface), environment variables
- **link time:** DLLs, class loading
- **run time:** virtual functions, inheritance & polymorphism, factory-based instance creation, delegation

Binding Time Consequences

	flexibility	performance	code size	complexity
source time	-	+	+	-
compile time	+	+	+	-
link time	+	+	+	-
load time	++	+	+	+
run time	+++	-	-	+

Relationship to MDS

- **MDS does not define the process** of how to come up with all these issues (domain analysis, variability analysis, etc.), although I think this is important!
- MDS also does **not explicitly talk about binding times**. However, it implicitly assumes that features are bound statically during subsequent model transformations. MDA is very code (and model) generation centered.

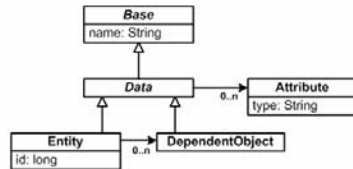
Feature models can thus be used to **model which feature will be fixed at which stage** of the model-transformation sequence.

- Generators **cannot just generate code**, also configuration files, make files, linker specs, ...
- You can **move stuff** between the platform and the generated code
- **A generator is the unifying tool for PLE**

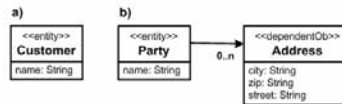
Variants and Models

- **Structural Variations**

Example Metamodel



- Based on this sample metamodel, you can build a **wide variety of models:**



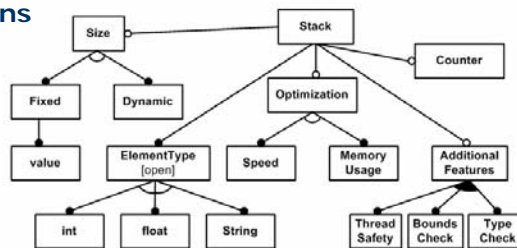
- **Non-Structural Variations**

Example Feature Models

Dynamic Size, ElementType: int, Counter, Threadsafe

Static Size (20), ElementType: String

Dynamic Size, Speed-Optimized, Bounds Check



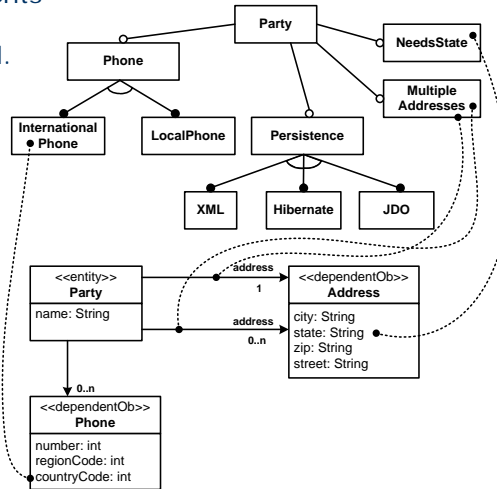
Variants and Models II

- It is especially useful to **combine structural and non-structural** variations
 - specifically, you may want to „configure“ structural models with the help of feature models,
 - we want to describe variants of structural models (and use these variants as generator input)
- Examples:
 - A party may have one or more addresses
 - A party may store telecontacts or not
 - In case of telephone numbers, you may want to store the country code
 - Addresses may have the *state* field (USA)



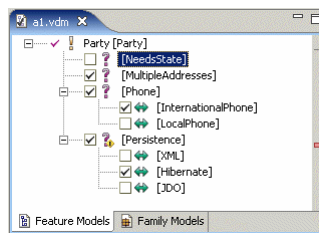
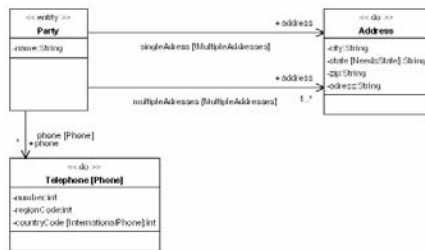
Variants and Models III

- You can **assign** model elements of the structural model to features in the feature model.
- The respective model elements are only there if the associated feature is selected,
- And it's removed, if the feature is not there.



Variants and Models III: Example Implementations

- pure-systems, Poseidon, openArchitectureWare



- Another very nice implementation is done by **Krzysztof Czarnecki and Michal Antkiewicz** at the University of Waterloo based on
 - Eclipse
 - Rational Software Modeller
 - Their own feature modelling Plugin

