

Modeling Variability

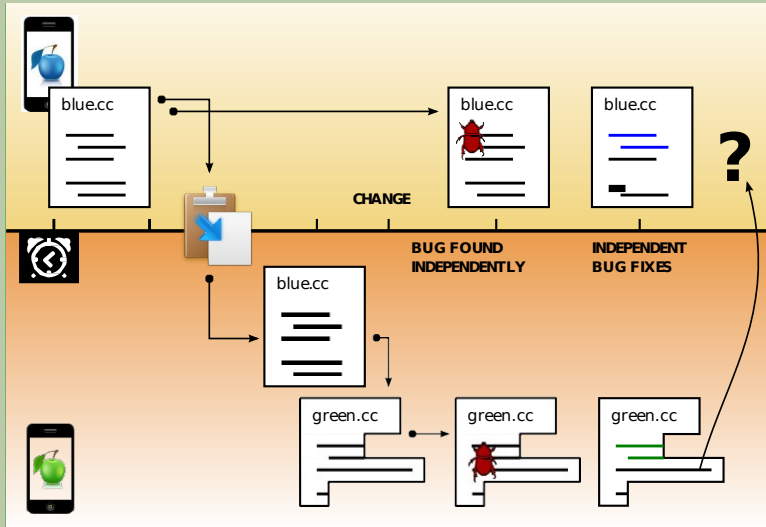
Andrzej Wąsowski

... and some notes by Hans Vangheluwe

IT UNIVERSITY OF COPENHAGEN

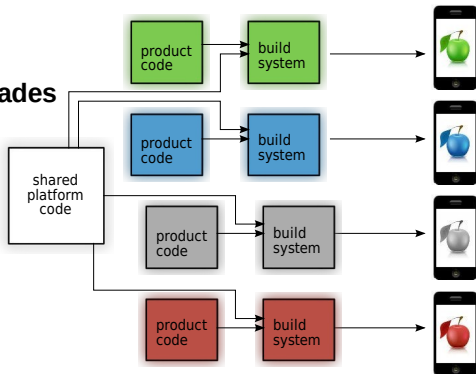
PROCESS AND SYSTEM MODELS GROUP

family of software products: “Clone-And-Own”



Opportunistic Reuse Does Not Work

- ▲ Common scenario:
version the code, **reuse** when **opportunity** appears
- ▲ If the file to be reused needs change, **copy** it
You clone-and-own it
- ▲ **Benefit** from **quickly** available functionality
- ▲ But have to test, debug, change and evolve the file **yourself**
- ▲ **Product specific** code grows
- ▲ **Platform** code **diminishes and degrades**

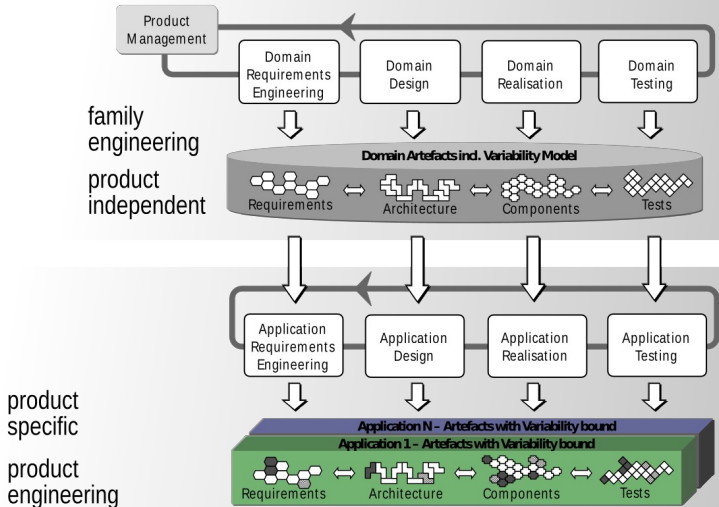


SUCCESSFUL REUSE IS

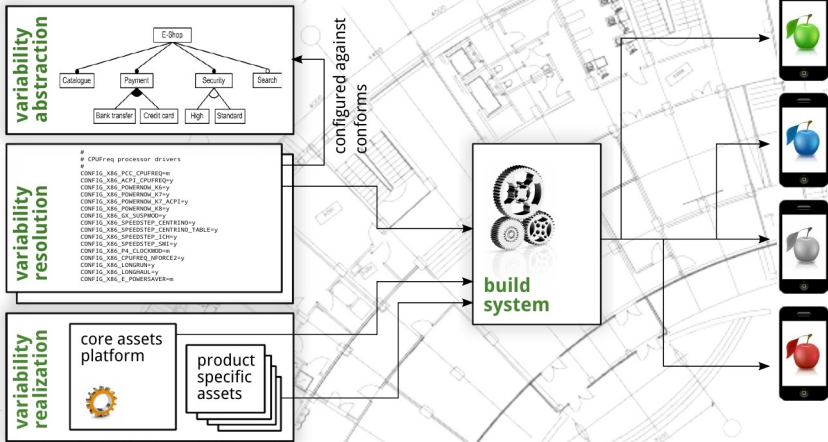
**PROACTIVE
PLANNED
MANAGED**

Domain vs Application Processes

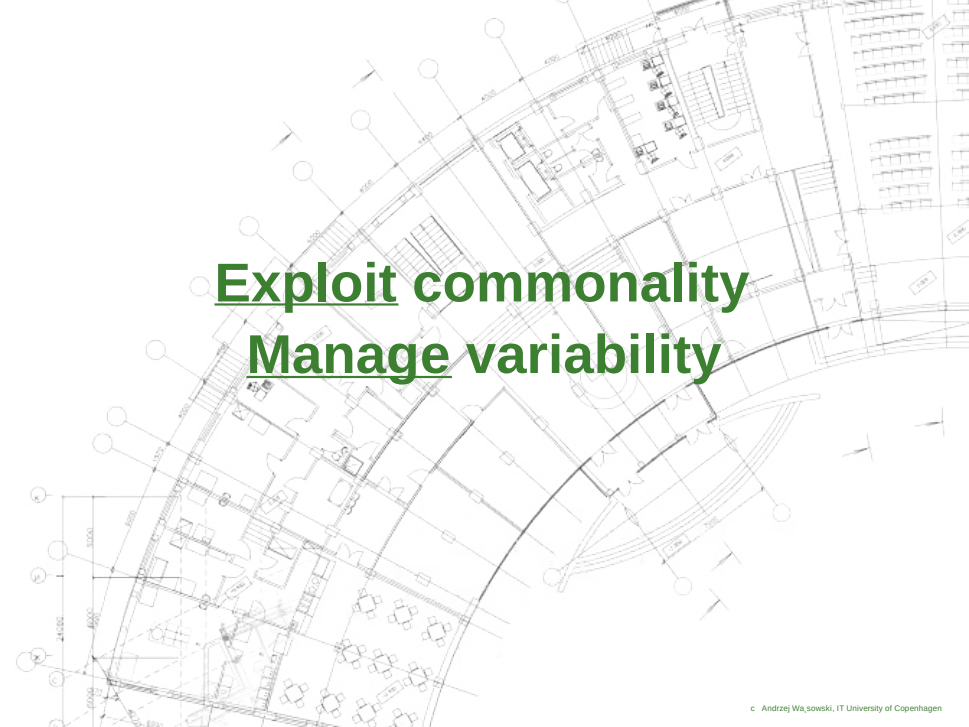
Pohl et. al. Software Product Line Engineering



A Simple Product Line Architecture

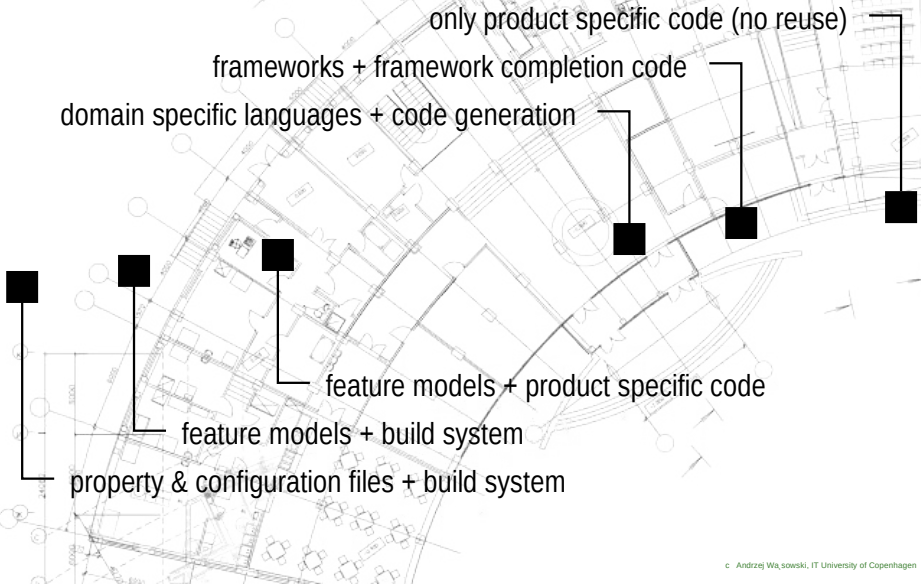


- ▶ Less product specific = more reuse: development/tests/debugging/build
- ▶ Model of *commonality* and *variability*.
- ▶ Scope under control. Explicit feature life cycle

The background is a detailed architectural floor plan of a building. It shows various rooms, corridors, and structural elements. There are numerous dimension lines with numerical values (e.g., 4000, 3000, 24500) and circular markers with letters (e.g., K, P, C) indicating specific points or sections. The plan is oriented diagonally.

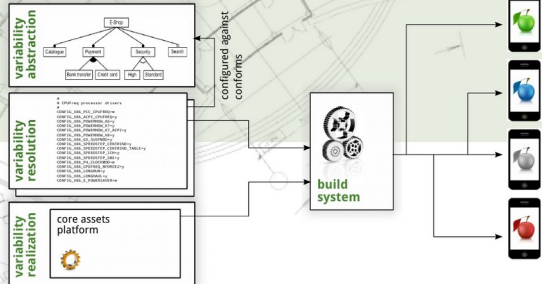
Exploit commonality
Manage variability

Spectrum of Variability Architectures



Implementation Technologies

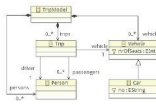
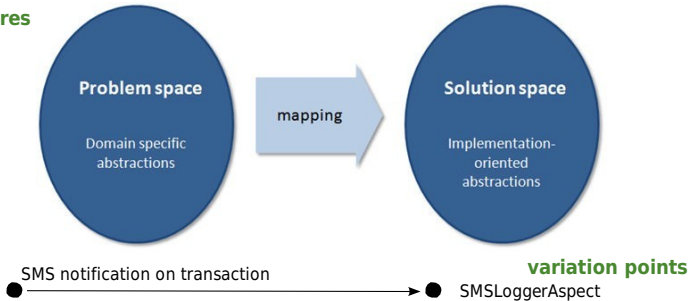
- ▶ Variability abstraction: FMs, DSLs, or **none**
- ▶ Variability resolution:
 - XML property file
 - FM configuration
 - Domain specific model (DSM)
- ▶ Variability realization:
 - general purpose code
 - w/ variability techniques
 - code generators
 - model transformers
 - parts may use DSLs
 - etc.



Problem Space

Solution Space

features



Feature Modeling (I)

feature:

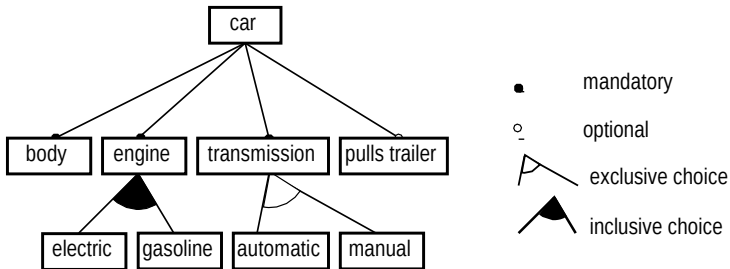
a single variability **increment** in the **problem domain** (~decision)

variation point:

a single variability **increment** in the **solution space**

Feature Modeling (II)

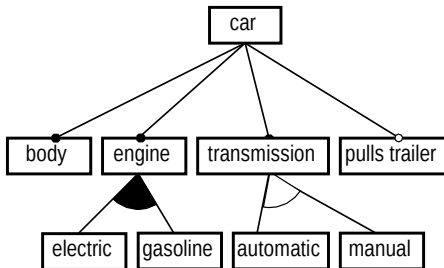
Example from Czarnecki'02



- △ **Hierarchy** constraints, for example:
 - manual requires transmission (each child node requires its parent node)
- △ **Groups** constraints: engine is electric or gas driven or both
- △ Not all constraints in hierarchy & groups, cross-tree constraints in text:
 - electric *requires* automatic
- △ Attributes are added like to classes (eg. engine volume)

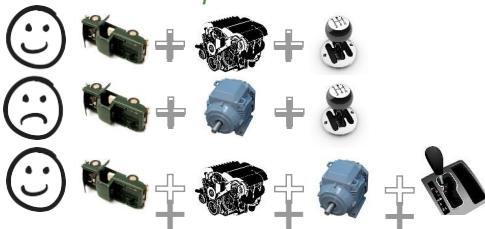
Feature Modeling (III)

Configuration



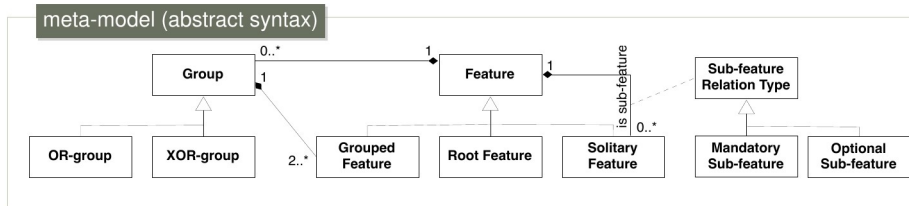
- mandatory
- optional
- ⌞ exclusive choice
- ▲ inclusive choice

electric requires automatic



Feature Models (IV)

An example meta-model from Janota'08



- Note a single generic kind of relations: **subfeature**
- No distinction between **kind-of** (inheritance) and **part-of** (containment), like class modeling does
- A characteristic feature of **configuration and constraint languages** (as opposed to structural modeling languages)
- Clafer (as a structural modeling language) supports the distinction, but so do other feature modeling languages

Technical Report

**CMU/SEI-90-TR-21
ESD-90-TR-222**



AD-A235 785

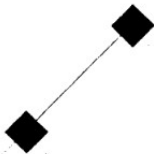


Carnegie-Mellon University
Software Engineering Institute

DTIC
S **ELECTE** **D**
MAY 24 1991
C

**Feature-Oriented Domain Analysis (FODA)
Feasibility Study**

**Kyo C. Kang
Sholom G. Cohen
James A. Hess
William E. Novak
A. Spencer Peterson
November 1990**



Rafael Capilla
Jan Bosch
Kyo-Chul Kang *Editors*

Systems and Software Variability Management

Concepts, Tools and Experiences

 Springer

Kang, K.C., Lee, H. (2013). **Variability Modeling.**

In: Capilla, R., Bosch, J., Kang, K.C. (eds)

Systems and Software Variability Management. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-36583-6_2

commonality and **variability** (C&V)

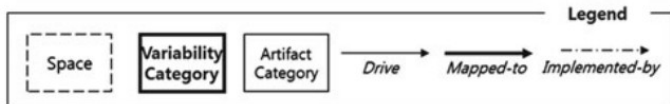
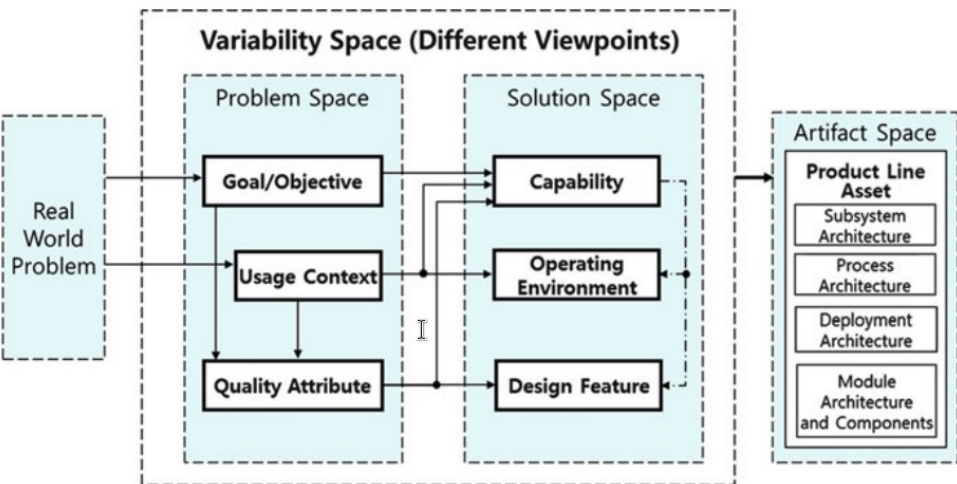
product **features** =
“stakeholder visible characteristics of products”

Feature Oriented Domain Analysis (FODA)

feature models: features are **organized** using
“consists of” and “generalization/specialization” relationships
(extensions with XOR etc. exist)

problem domain vs. **solution** domain (is relative)

basis for: formal analysis of feature model, feature configuration, generative programming, ...



Software Product Line engineering (SPL)

Elevator Control System (ECS) product line

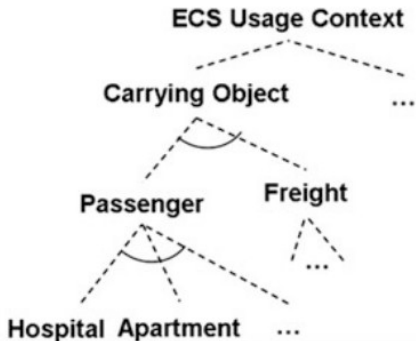
a

Goals/Objectives



b

Usage Contexts



Legend

Mandatory



Optional

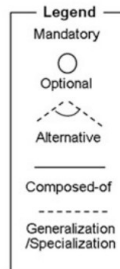
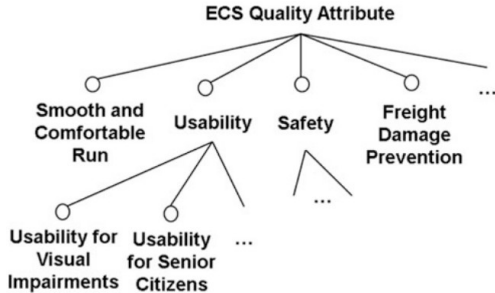


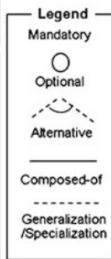
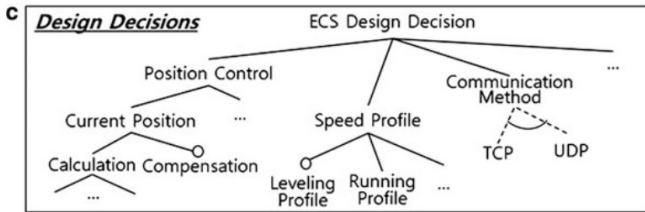
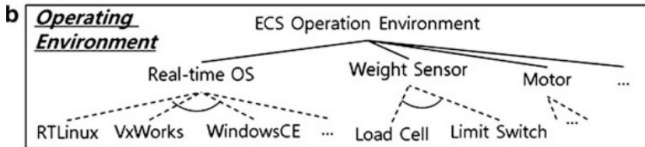
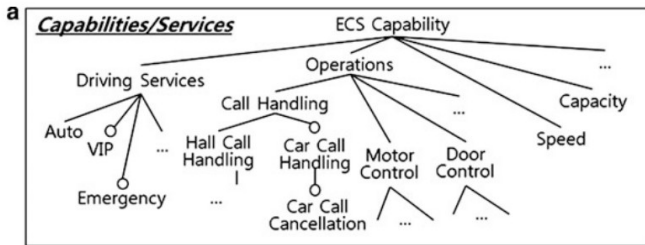
Alternative

Composed-of

Generalization /Specialization

problem space feature models of the ECS product line





solution space feature model of the ECS product line

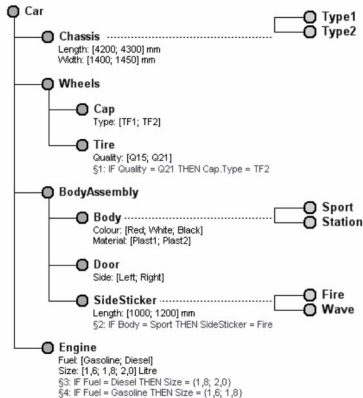
Feature Binding Time

A feature can be **included** in the asset or a product at any product line lifecycle phase, but their **availability** for use can be determined at the time of inclusion or at any time after by **enabling** or **disabling** the included feature.

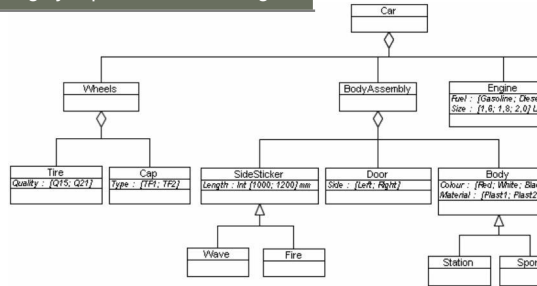
Feature Modeling vs Class Modeling

A feature model in *Product Variant Master Notation* (Hvam)

A feature model



A roughly equivalent class diagram



More on this: Bałk. Czarnecki. Wałsowski. *Feature and Meta-Models in Clafer: Mixed, Specialized, and Coupled*. SLE 2010

Above models from: Haug. Degn. Poulsen. Hvam. *Creating a documentation system to support the development and maintenance of product configuration systems*

Applications of Feature Models

Design & Management



domain
modeling

product line scoping
product line mngmt

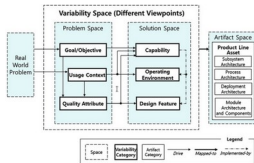


code generation
driving build system

driving
testing



Development & Test



How To Build Feature Models?

Two strategies...

top-down

- ▲ **Big-bang** adoption
- ▲ Perform **careful domain analysis**
- ▲ Document **concepts, abstractions and relations** between them in a FM

bottom-up

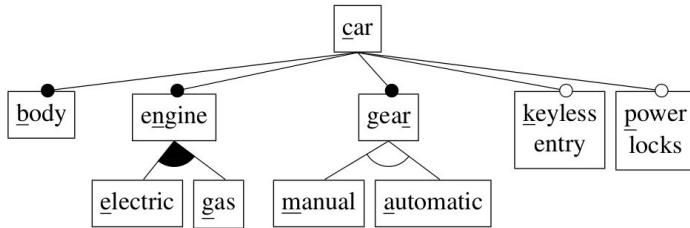
- ▲ Identify a **cloned component**
- ▲ Find the **patches** that describe differences
- ▲ Translate **diffs to variation points**
- ▲ Organize **variation points into features**, and a hierarchy
- ▲ Works well with **incremental adoption**
- ▲

Variability Modeling is The Success Story of Modeling





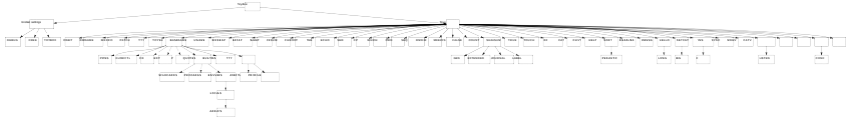
A Laboratory Feature Model



keyless_entry → power_locks

A Healthy Wild Feature Model Cub

ToyBox project, 71 features



**The Linux Kernel has 6-12K features, depending how you count!
But maximum depth is 8, most leaves are at 4!**

↓ this is the Linux kernel model fit to the slide width ↓

Is FODA special? Not really!

eCos configurator

The screenshot displays the eCos configurator interface. On the left is a tree view of configuration options, and on the right is a table of selected properties and their values.

Tree View (Left):

- ROM filesystem (v3_0)
- JFFS2 filesystem (v3_0)
 - ☐ Support garbage-collection background thread
 - ☒ Include write support for JFFS2
 - ☐ Support for NAND flash
 - Debug level (0)
 - ☒ Compress data
 - ☒ Compress data using zlib
 - ☒ Compress data using rtme
 - ☐ Compress data using rubin
 - Set the default compression mode (PRIORITY)
 - Memory pool size (0)
 - Additional compiler flags (-D_ECOS)
 - Suppressed compiler flags
 - Support for fileio's struct dirent d_type field
- JFFS2 FS tests (tests/jffs2)
- Linux compatibility layer (v3_0)
- Zlib compress and decompress package (v3_0)
- VNC Server (v3_0)

Property Table (Right):

Item	Conflict
CYGPKG_POSIX_CLOCKS	Unsatisfied R
CYGHWR_IO_FLASH_DEVICE	Unsatisfied R
CYGPKG_FILEIO_FNMATCH	Unsatisfied R

Property	Value
URL	ref/fileio.html
Macro	CYGOPT_FS_JFFS2_COMPRESS
File	/home/shshe/Dev/ecos/ecos-3.0/
Enabled	True
Flavor	bool
Define	JFFS2_COMPRESSION
DefaultValue	1

Compression and decompression are entirely handled by the file system and are fully transparent to applications. However, selecting this option increases the amount of RAM required and slows down read and write operations considerably if you have a slow CPU.

- ▲ **Linux kernel and eCos operating system** use similar configurator, controlled by **textual variability models**
- ▲ **Trees become unwieldy** very fast
- ▲ Many tools used **linearized trees**, like above
- ▲ Nice **trees are good for PowerPoint**, whiteboard and brainstorming

It is easy to design a textual syntax

Kconfig of Linux kernel, designed by non-experts, with no tools

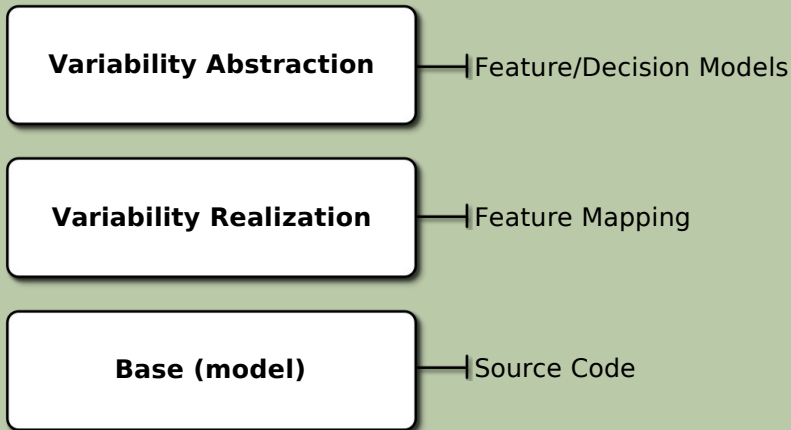
CDL of eCos, designed by non-experts, with no tools

```
k-1 menuconfig MISC_FILESYSTEMS
k-2     bool "Miscellaneous filesystems"
k-3
k-4     if MISC_FILESYSTEMS
k-5
k-6         config JFFS2_FS
k-7             tristate "Journalling Flash File System" if MTD
k-8             select CRC32 if MTD
k-9
k-10
k-11
k-12
k-13     config JFFS2_FS_DEBUG
k-14         int "JFFS2 Debug level (0=quiet, 2=noisy)"
k-15         depends on JFFS2_FS
k-16         default 0
k-17         range 0 2
k-18         --- help ---
k-19             Debug verbosity of ...
k-20
k-21
k-22     config JFFS2_FS_WRITEBUFFER
k-23         bool
k-24         depends on JFFS2_FS
k-25         default HAS_IOMEM
k-26
```

```
c-1 cdl_component MISC_FILESYSTEMS {
c-2     display      "Miscellaneous filesystems"
c-3     flavor       none
c-4 }
c-5
c-6 cdl_package CYGPKG_FS_JFFS2 {
c-7     display      "Journalling Flash File System"
c-8     requires     CYGPKG_CRC
c-9     implements  CYGINT_IO_FILEIO
c-10    parent       MISC_FILESYSTEMS
c-11    active_if    MTD
c-12
c-13    cdl_option CYGOPT_FS_JFFS2_DEBUG {
c-14        display      "Debug level"
c-15        flavor       data
c-16        default_value 0
c-17        legal_values 0 to 2
c-18        define       CONFIG_JFFS2_FS_DEBUG
c-19        description  "Debug verbosity of...."
c-20    }
c-21
c-22    cdl_option CYGOPT_FS_JFFS2_NAND {
c-23        flavor       bool
c-24        define       CONFIG_JFFS2_FS_WRITEBUFFER
c-25        calculated   HAS_IOMEM
c-26    }
```

CVL Architecture for Dummies

The degree of coupling can be controlled by moving the mapping



CVL Architecture for Linux

Variability Abstraction

| KConfig files

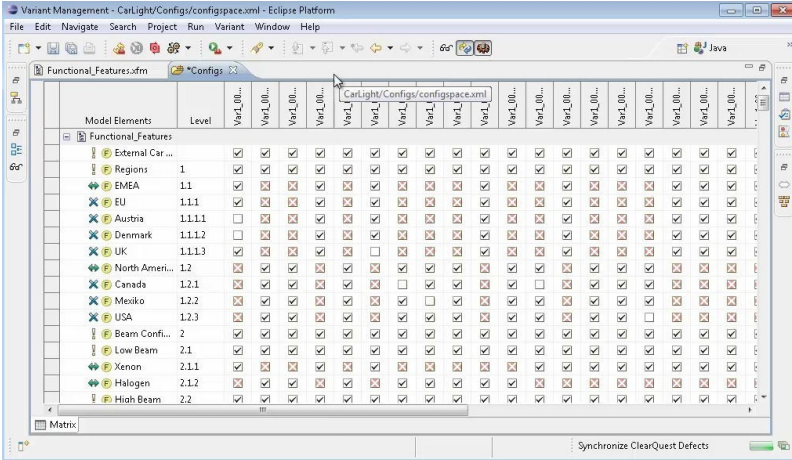
Variability Realization

| KBuild files + CPP

Base (model)

| C-code

Grid View in PureVariants

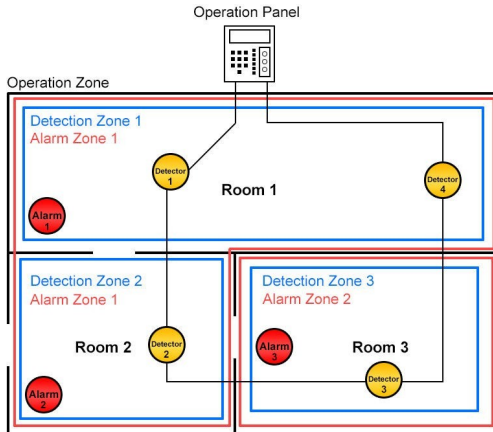


<https://www.ptc.com/en/products/pure-variants>

- ▲ Commercial tools support **multiple views** of the same model
- ▲ Some vendors: Pure Systems (DE), Big Lever (US), most PLM tools
- ▲ Clafer also has the grid view

Are all variability models trees?

A Fire Alarm System



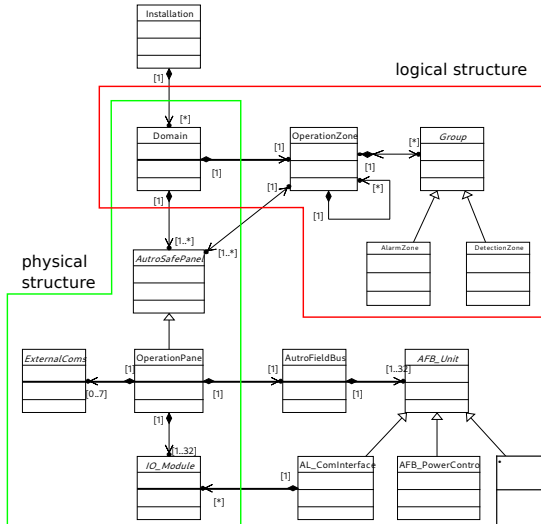
- ▲ detection zones
- ▲ alarm zones
- ▲ wiring
- ▲ three different structures
- ▲ Modeled as constrained class diagrams!
- ▲ Sometimes called **topological variability**

Berger. Stanculescu. Øgård. Haugen. Larsen. Waśowski. *To connect or not to connect: experiences from modeling topological variability*. SPLC 2014

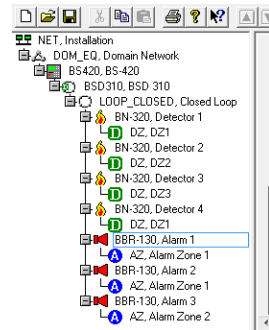
Fantechi. *Topologically configurable systems as product families*. SPLC 2013

Modeled using class diagrams in Papyrus

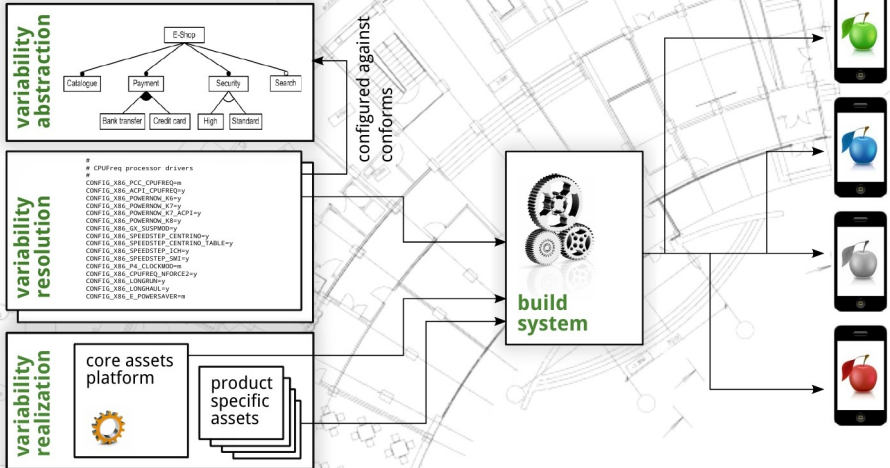
A model view showing two model hierarchies



A home grown **configurator** used to instantiate models



Connect Abstraction to Realization



Spectrum of Variability Architectures

Stay as close to the left as possible

only product specific code (no reuse)

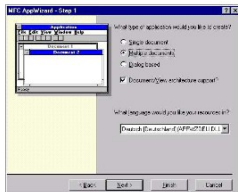
frameworks + framework completion code

domain specific languages + code generation

feature models + product specific code

feature models + build system

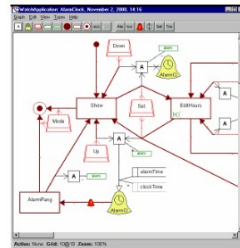
property & configuration files + build system



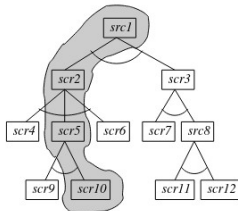
Wizard



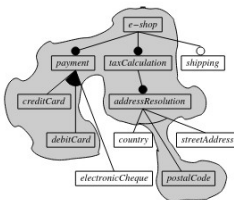
Feature-based configuration



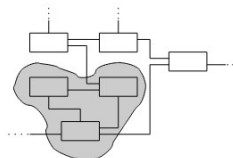
Graph-like language



Path through decision tree

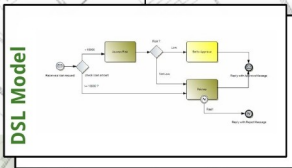
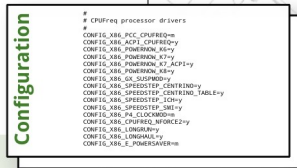
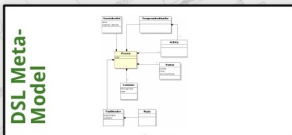
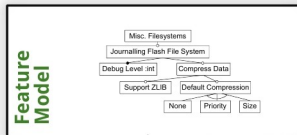


Subtree of feature model



Subgraph of (infinite) graph

Feature Models vs DSLs



conforms to

conforms to

- ▶ Feature models are **ready and simple** (no design effort, deep insight)
- ▶ DSL requires **design effort**, but rewards with more **expressiveness**
- ▶ Effort also translates to **maintenance**
- ▶ FM effort is offset by **existing feature modeling tools**
- ▶ DSL development effort is offset by **language workbenches**

Advice on Realization

[Stahl and Völter]

- ▲ Choose **functional** domain **concepts** as features / DSL concepts
- ▲ Start a **small** domain model and **grow** it **iteratively**
- ▲ Keep the **build automatic** at all times
- ▲ Generate/synthesize **legible code/models**

Generative Programming

Methods,

Tools, and

Applications



Krzysztof Czarnecki
Ulrich W. Eisenecker

 **WILEY**

TIMELY. PRACTICAL. RELIABLE.

Model-Driven Software Development

Technology,
Engineering,
Management

Thomas Stahl,
Markus Völter

with Jörn Bettin, Arno Haase
and Simon Helsen

Foreword by Krzysztof Czarnecki

Translated by Bettina von Stockfeth



KNOWLEDGE-BASED CONFIGURATION

FROM RESEARCH TO BUSINESS CASES



MK
MANAGEMENT KNOWLEDGE

EDITED BY
ALEXANDER FELFERNIG, LOTHAR HOTZ,
CLAIRE BAGLEY AND JUHA TIHONEN

Udbergrettet skrevet af Morten
LARS HVAM
NIELS HENRIK MORTENSEN
JESPER RIIS

Product Customization



 Springer

Udbergrettet skrevet af Morten