Variability Modeling
by Øystein Haugen

Version 141107
What do we mean by “variability”?  

- **Product Line variance**  
  - often variants of the same software base

- **Cross-cutting variability**  
  - often variability is orthogonal to the software design  
  - variability needs are discovered after the first software design

- The variability designer is not always the software designer  
  - division of labor and of competences

11/7/2014 CVL – Common Variability Language
### Common ways to model variability

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Framework/Configuration</th>
<th>Union-of-all-systems</th>
<th>Domain Specific Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>How?</td>
<td>By mechanisms of a general language</td>
<td>As annotations to a language</td>
<td>By the specific language mechanisms</td>
</tr>
<tr>
<td>Unforeseen modeling needs</td>
<td>Just enhance the final model</td>
<td>Enhance the product line model</td>
<td>If not expressible, enhance the language</td>
</tr>
<tr>
<td></td>
<td><strong>Function, Type, Inheritance, Template, Plugin</strong></td>
<td><strong>Pragma, Stereotype, feature diagram</strong></td>
<td></td>
</tr>
</tbody>
</table>

**INF 5150**

11/7/2014 CVL – Common Variability Language
The worlds of UML and DSL

- **UML**
  - 700 pages, >100 classes
  - 2000 issues after UML 2.0
  - semantic variation points
  - No correct implementation (?)

- **Education**
  - Taught in schools
  - Used by thousands
    - misused by even more

- **Migration**
  - Big steps (UML 1 to UML 2)
  - Up front review process

- **DSL (imagine one you know)**
  - no pages, no definition doc.
  - <50 classes
  - The code generator is the semantics

- **Education**
  - Ad hoc teaching in-house
  - Used by a few dedicated ones
    - but hardly misused

- **Migration**
  - Many small steps
    - and version problems
  - Review process post festum
UML and DSL continued

- **UML problem**
  - there are many ways to goal
    - all of them to be implemented
  - typically something is skipped

- **UML Profiles**
  - additions to UML
    - carries UML on its back
    - with all its faults (in principle)
    - and its benefits
  - thus even bigger than UML
    - needs to piggyback on UML tool

- **DSL problem**
  - there is no way to the goal
    - and a way must be made
  - typically need for abstractions

- **Meta-libraries**
  - additions to the DSL
    - selected for the needs
    - but the concepts must be combined with the DSL
  - libraries have useful stuff
    - and also useless stuff
    - clutters the DSL concepts
The Compromise

- focused on domain
- easy code generator
- standardized
- available general tools

DSL

VAR

DSL+VAR
ARI – the simplest DSL

```
3*x^2 - 4*x + 1
3*x*y + 5
```
The need for abstraction

- The ARI language can describe any arithmetic expression
  - thus it is "complete" within its domain
  - but modeling every desirable arithmetic expression from scratch may be too time consuming or too error prone

- We would like to use some of the arithmetic expressions that we have already made
  - or we would like to have the chance to define general patterns and reuse those effectively in new descriptions

- Two ways to achieve simple abstraction from ARI:
  - Enhance the DSL by some general function or type concept
  - Apply a generative approach: The VARiability language
FUNC: enhancing ARI with function

$f(x) = 3 \times x^2 - 4 \times x + 1$
$g(y) = f(y) \times R$
$h(x,y) = 3 \times x \times y + 5$
The generative ARI+VAR+RES

DSL model
(ARI: 3\*x^2 − 4\*x + 1)
One example ARI+VAR+RES

one formula

-6x^2-23x+14: ResolutionModel

-6: ValueResolution

varyingcoeffs: ValueResolution

23: ValueResolution

14: ValueResolution

3A^2-4Bx+1C

any formula

Acoeff: VariationElement

Bcoeff: VariationElement

Ccoeff: VariationElement

A: Value Transformer

B: Value Transformer

C: Value Transformer

one formula

3x^2-4x+1: expr

3x^2: aoperand

-4x: aoperand

1: aoperand

3: moperand

x^2: moperand

4: moperand

x: moperand

3: num

x: var

2: num

4: num

-6x^2-23x+14

3x^2-4x+1
Across the river for water?

- We started with a specific formula (arithmetic expression)
  - and ended up with a specific second order polynomial

- What is gained?
  - The basic DSL must always have the full modeling capabilities
    - thus adding variability does not add the ability to model new products
  - Modeling is not only about what can be described
    - then we could all settle for assembler code
  - Modeling is about effective description
    - cooperation, maintenance, similarities

- ARI can express every polynomial
  - but ARI cannot express the concept "Polynomial"
Comparing FUNC and ARI+VAR+RES

- **FUNC**
  - Functions vary over values at runtime
  - Adding functions to ARI adds obligation to implement them in the code generator
  - Adding functions to ARI is a job for the language designer
  - Functions represent variability aligned with the original design
  - Adding functions to ARI does not exclude also applying VAR

- **ARI+VAR+RES**
  - VAR expressed values that vary at compiletime
  - Applying VAR to ARI creates no new implementation obligations
  - Applying VAR to ARI is a job for the variability designer
  - Applying VAR may cross-cut the original design
  - Applying VAR to ARI does not exclude adding functions to ARI
Advantages of the BVR approach

- Base Language (DSL)
  - Focused language design for the DSL designers
  - Simplest implementation of DSL tools
- Variation and Resolution Languages
  - Standard
    - Up front reviewing process
    - Community
    - Common variability tools
  - Requirements
    - Only that all the metamodels are described in MOF
      - which will make the semantics of the VAR language definable in QVT
- Language Engineering Process
  - Division of labor and reaping differences of competence
Goals for OMG Standardization

- Intuitive ways to describe the product line variability on high abstraction level
  - and means to configure products without being a software specialist
- Automatic means to produce products from product line (cf. MDA)
- Generic ways to describe variability
  - that can work well with product line descriptions in any base language
- Techniques for making generic tools that can work well with the tools for the base languages
Common Variability Language

- CVL = Common Variability Language
- The objective of CVL is to enable the specification of the variability in product line models in order to support seamless product line modeling across the whole product line engineering process.
- CVL is a specification language including a metamodel, semantics and concrete syntax for variability specification.
- Variability specifications shall relate to a base product line model that describes the whole product line and shall comprise:
  - a variability model with the following elements: a model of possible choices and relationships between those choices and the base model
  - resolution models which resolve variability (by a set of choices) and thus define specific product models.
- CVL shall support base models in languages that are defined by means of metamodels, including UML and Domain Specific Languages.
CVL Tutorial

CVL in a Nutshell
CVL overview and terms

- **CVL**: Generic & Standardized
- **DSL**: Focused on a domain

- **Variability model**: Specification in CVL of base model variabilities
- **Base model**: Product line model in any MOF-compliant language
- **Execute CVL**: Product models fully described in the base language.
- **Resolved models**: All regular base language tools can be applied to these models
- **Resolution models**: Selection of a set of choices in the variation model
Variation Points over base model

Variability in this example:

- Part EmergencySupply is optional
- Part HighSpeedConnector is optional
- Port EmgPowerCtrl on block Printer is optional
- Value of attribute threshold in block EmergencyPower is variable
Variation points in CVL

- Variation Points refer to Base objects
- Variation Points define the base model modifications precisely
- There are different kinds of Variation Points
  - Existence
  - Value assignment
  - Substitution
  - Opaque variation point
  - Configurable Unit
VSpec trees and binding

VSpec tree
- choice
- variable
- constraint

Variation points

11/7/2014 CVL – Common Variability Language
VSpecs in CVL

- VSpecs (Variation Specifications) describe the abstract variability
- Every Variation Point is bound to exactly one VSpec
- VSpecs come in different kinds:
  - Choice
  - Variable
  - Constraint
  - VClassifier
  - CVSpec
Constraints in CVL

- CVL include a basic language for expressing constraints on the VSpec tree
  - Propositional logic is supported
- CVL also has the opportunity to let you apply other constraint languages like OCL
Resolution

Resolution model
- False
- True
- 90

Variation points
- HighSpeed
- threshold > 100
- EmergencyPower

Attributes
- threshold:int

Operations
- powerCtrl

:ObjectExistence
- MainPower
- EmgPower
- inputSection
- highSpeedConnector
Variability Resolution in CVL

- VSpecResolution elements refer to VSpecs
- The set of valid Resolutions is restricted by the constraints
- Represent information necessary to materialize product models
  - Actual yes/no decisions on Choices
  - Actual values to Variables
  - Instances of VClassifiers
  - Configurations of CVSspecs/Configurable Units
Materialization

Printer

HighSpeed & threshold>100 → EmergencyPower

Resolution model
False
True
90

Variation points
Miss
HighSpeed & threshold>100
EmergencyPower

Materialization

threshold: int

Materialization

threshold=90
Materialization (with annotations)

Product Line Model

Product Model (materialized model)
Configurable Unit and VInterface
CVL Architecture

- Configurable Units
- Variability Interfaces
- Variation Points
- Variability Realization
- VSpecs
- Constraints
- Variability Abstraction
- Resolutions
- Base Model

11/7/2014 CVL – Common Variability Language
CVL Tutorial

CVL in Detail
Recall CVL Architecture

Configurable Units ➔ Variability Interfaces

Variation Points ➔ Variability Realization

VSpecs ➔ Constraints

Variability Abstraction ➔ Resolutions
Variability Specification (Vspec)

- **Scanner**
  - Choice
  - BothSides
  - Type
    - Group multiplicity of type XOR/alternative (1..1): a scanner can be either of type color or BW
    - Is implied by parent=false (optional)
    - Is implied by parent=true (mandatory)
    - speed: integer
    - Choice
      - Color
      - BW

- Variable
Variability Specification (Vspec)

- Abstract variability specifiers
- Similar to features in feature modeling
- Essentially **decisions** like in decision modeling
- Example: “GPS” is a feature of a camera, but it is also a choice, which can be decided yes/no
- VSpecs can be used for feature modeling
- Variation points are bound to VSpecs, giving semantics to VSpecs.
- VSpecs are arranged in trees,
- Parent-child relations organize the resolution space in usual way, like in FMs.
VSpec Types (continued)

- **Choice** requires yes/no decision
- **Variable** is a VSpec allows for providing a value of a specified type
- **VClassifier** allows for creating instances and then providing per-instance resolutions for the VSpecs in its sub-tree
- **Composite VSpecs** – used for modularity (not covered)
VClassifier

Instance multiplicity of Printer indicating 0 to any number of instances

Every printer has a Speed choice

VClassifier

PrinterPool

Printer [*]

Color

Speed

1..1

X20

X22
Constraints

- Express intricate relations between VSpecs
- That cannot be captured by trees

- CVL offers basic constraint language
- A sublanguage of OCL

- Other constraint languages, including full OCL, are admitted.
- We show the basic language by example
- And a bit of the full OCL
Simple Propositional Constraints

PrinterPool

fax
printer

fax implies printer
Complex Propositional Constraints

(fax implies printer) and (copier implies (scan and printer))

- Global context used in this example
- Using PrinterPool would be equivalent
Arithmetic Constraints

PrinterPool

minSpeed : int
speed : int

Speed = minSpeed + 300
Path Expressions

Now using fully qualified names, instead of relying on name disambiguation.
Implicit Quantification

PrinterPool

Printer [1..*]

color    colorPrinterHead

color implies colorPrinterHead
Recall CVL Architecture

- Configurable Units
- Variability Interfaces
- Variation Points
- Variability Realization
- VSpecs
- Constraints
- Variability Abstraction
- Resolutions
- Base Model
The Base Model of the Scanner

- **ParametricSlot Assignment**
  - f:Feeder speed=100
  - q:Quality

- **Object Substitution**
  - t:Turner

- **Object Existence**
  - Quality
    - BW
    - Color
  - Feeder speed:integer
  - Turner
The VSpec layer of the Scanner

Diagram:

- **Scanner**
  - **BothSides**
    - **Type**
      - **Speed**: integer
      - **Color**
      - **BW**
One resolution of the Scanner

Scanner

Scanner = True

Type = True

speed = 200

BothSides = False

Color = True

BW = False

speed: integer

BothSides

1..1

Type

Color

BW
Realization Layer of the Scanner (Object)

Scanner

f: Feeder
  speed: integer
  speed=100

t: Turner

q: Quality

ParametricSlotAssignment
  slotIdentifier="speed"

ObjectExistence

Feeder
  speed: integer

Turner

Quality
  Color
  BW

Scanner
  speed: integer

BothSides
  Type
  1..1

Color

Object Substitution

BW

Object Substitution
The resulting product of the Scanner

Scanner

f: Feeder
  speed=200

q: Color

Feeder
  speed: integer

Turner

Color

BW

Scanner = True

speed = 200

BothSides = False

Type = True

Color = True

BW = False
The Printer Pool

VClassifiers
The base model of the PrinterPool

PrinterPool

p:Printer[1..*]

Printer

color:Boolean

speed:String

PX
The VSpec layer of the PrinterPool

```
PrinterPool

Printer [*]

Color

Speed

X20 X22

1..1
```
One resolution of PrinterPool

PrinterPool

Printer [*]

Speed

Color

VInst1:Printer

VInst2:Printer

PrinterPool = True

Speed = True

Color = True

X20 = True

X22 = True

Speed = False

Color = False

X20 = False

X22 = False

1..1
Realization Layer of the PrinterPool

PrinterPool

p:Printer[1..*]

Printer
color:Boolean
speed:String

PX

PrinterPool

PrinterPool

Printer [*]

Color

Speed

placement
replacement

(replacement of Color not shown)

SlotAssignment
slotIdentifier="speed", value="X20"

SlotAssignment
slotIdentifier="speed", value="X22"

1..1

X20

X22
Achieving generality: The Fragment

- CVL has two layers: abstraction layer and realization layer
- Substitution is the key concept of realization layer
- The fragments are on the model instances and they are all defined by MOF
Resulting Product model of PrinterPool

one $P1$ and one $P2$ are included in $p:Printer$.

How to achieve this is not shown here for simplicity.
Semantics of Fragment substitution

- Simple fragment substitution when multi=False
  - Delete the placement
  - Make a copy of the replacement
  - Bind the replacement copy to the hole of the placement
    - The boundary points must correspond

- Multiple fragment substitution when multi=True
  - Delete the placement
  - Make a number of copies of the replacement
    - the actual number of copies is given by the resolution model resolving the VSpec that refers the given FragmentSubstitution
  - Bind all the copies to the hole of the placement
    - All references into the placement must have multiplicity more than one
Opaque Variation Point (1/2)

<table>
<thead>
<tr>
<th>MainPower</th>
</tr>
</thead>
<tbody>
<tr>
<td>- defaultAttribute: String</td>
</tr>
<tr>
<td>- powerType: String</td>
</tr>
</tbody>
</table>

Diagram shows a UML class diagram with classes `MainPower`, `Adaptor`, `EmergencyPower`, `acmePower`, and `TexasPower`. The diagram illustrates dependencies and associations between these classes.
Opaque Variation Point (2/2)

Sample Transformation Rule for
1. Copying all attributes associated with self->placeHolder to self->sourceObject
2. Assign self->sourceObject.powerType = self->sourceObject.Name
3. Substitute self->placeHolder by self->sourceObject

Abstraction (vSpec Tree)

Realization Variation Point

OpaqueVariationPoint 1

OpaqueVariationPoint 2

Base Models

SemanticSpec1

SpecialSubstitution

type

spec
Recall CVL Architecture

Configurable Units → Variability Interfaces

Variation Points → VSpecs → Constraints → Resolutions

Variability Realization → Variability Abstraction

Base Model
Configurable Units: Mission Statement

- Produce configurable components reusable across multiple projects
- Associate a collection of variability declarations with a base model container
- Containers: UML/SYSML component, package or class, etc ...
- Hide details, Expose variability interface
- Component can be configured
- Reuse CUs: clone-and-own or by reference
Examples

remember?
A Skoda Yeti car can have the following combination of features:

1. Manual and Diesel
A Skoda Yeti car can have the following combination of features:

2. Adventure and Benzin

<table>
<thead>
<tr>
<th>Correct</th>
<th>Wrong</th>
<th>Cannot be answered from model</th>
<th>I don’t know</th>
</tr>
</thead>
</table>
A Skoda Yeti car can have the following combination of features:

10. Automatic and Adventure and Benzin and 2-wheel-drive
Active and Diesel implies Manual
Active and Benzin implies 2-wheel-drive
Adventure implies Diesel and 4x4
Active and 4X4 implies Diesel
Active and 4X4 implies Manual
Active and Diesel implies Manual
Active and Benzin implies 2-wheel-drive
Adventure implies Diesel and 4x4
Active and 4X4 implies Diesel
Active and 4X4 implies Manual

A Skoda Yeti car can have the following combination of features:

2. Adventure and Benzin
Active and Diesel implies Manual
Active and Benzin implies 2-wheel-drive
Adventure implies Diesel and 4x4
Active and 4X4 implies Diesel
Active and 4X4 implies Manual
CVL – Early Experiments

SINTEF CVL Tool
(Projects CESAR and VERDE)
Example of industrial applications

Paper Cutting Machine

Elevator

Pumps

Pumps
Safe Power Drive System – PDS (SR)

- Plug-in safety module
- Drive
- Safe Drive
- STO (Safe Torque Off - cut power to motor)
- Optional tachometer (Speed and direction)
- Optional brake

Diagram details:
- Safety Module
- Motor Controller
- Encoder
- Brake
- Motor
- Digital I/O
- STO
- Com
- Power
- Optional
Domain and Product Variability

- **Conveyor belt** safety requirements:
  1. An emergency stop device shall be provided.
  2. The safety-related part of the control system shall comply with Safety Integrity level SIL2.
  3. Devices for detection of unexpected start-up shall be provided.
  4. The drive system shall be designed so that the speed does not exceed the designed speed.

- **Hoisting machine** (e.g. crane, elevator...) safety requirements:
  1. ....
  2. The safety-related part of the control system shall comply with Safety Integrity level SIL3.
  3. ....
  4. Machine shall be fitted with braking devices.

Safety functions selected:

- Safety Function 1: STO (Safe Torque Off) (mandatory)
- Safety Function 2: SSE (Safe Stop Emergency)
- Safety Function 3: SMS (Safe Maximum Speed)

Safety functions selected:

- Safety Function 1: STO (Safe Torque Off) (mandatory)
- Safety Function 2: SBC (Safe Brake Control)
- Safety Function 3: SSE (Safe Stop Emergency)
- Safety Function 4: SS1 (Safe Stop 1) (with encoder)
The Safety Module and a Safety Drive

Safety Drive Product Line

Safety Drive (one product with the Safety Module)

- Motor Controller
- Motor
- Safety Module (here simulated)

Product Configuration File

Executable Testing Code
Variability and domain design

Derivation

Deployment
The Executable Base Model: UML on Papyrus with profiles

Composite structure(s)

Test Component Motor Controller

Test Component Motor

Safety Module

Variability and domain design

Derivation

Deployment

Product model with tests

Executable Testing Code

Base Configuration Template

Configuration Fragments

Safety Module

Test Component

Motor Controller

Motor

Power Off

Communications

Configuration

Configuration

Set Target Speed

Power Off

Configuration

Configuration
Defining the Product Line with CVL Tool and Papyrus

CVL Abstraction Layer

CVL Realization Layer

Coloring the effects of the substitution
Variants of behavior: changes to a state machine

- **Placement**
- **Replacement**
- **Substitution**
Product Selection and Resolution (Hoisting Machine)
Execute CVL to obtain set of product models

- Execute CVL
  - on design time model
  - on configuration file model
UTP Test Cases integrated with model
Testing a product line

Variability and domain design

Derivation

Deployment

CONCEPTUAL VIEWS

The Product Line: CVL Abstraction Layer

Combinatorial Coverage arrays

CVL Resolution

CVL Realization Layer

The Product model with tests

Base DSL Model incl. tests

DSL metamodel (+ profiles)

DSL Fragments

Product Simulation Code

Executable JUnit Testing Code

Product Simulation Execution

generate

generate

compile and run

execute tests

select

CVL metamodel