Modeling 2: Variability Modeling in detail + Looking at the Yeti modeling experiment

Version 131004
Agenda for Modeling 2

- Variability Modeling with CVL in detail
  - Recap of feature modeling with VSpec trees in CVL (CVL Tool)
  - Resolution, Realization and Materialization
  - Configurable Units

- The Yeti Modeling Experiment
  - We look at the questions again – and understand what is right
Recall CVL Architecture

- Configurable Units
- Variability Interfaces
- Variation Points
  - Variability Realization
  - Variability Abstraction
    - VSpecs
    -Constraints
    - Resolutions
- Base Model
Variability Specification (Vspec)

- **Variable**
  - Group multiplicity of type XOR/alternative (1..1): a scanner can be either of type color or BW

- **Choice**
  - **Type**
  - Is implied by parent=true (mandatory)
  - Is implied by parent=false (optional)
  - **speed**: integer
  - **BothSides**
  - **Scanner**
  - **Color**
  - **BW**
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; explained later
Tree Structure Semantics

- Configuration semantics akin to cardinality based feature models
- Negative resolution implication
- Positive resolution implication
- Group Multiplicity
- Instance Multiplicity (classifiers)
The Yeti Modeling Experiment – walk through

We see through the questions and the modeling task.
A Skoda Yeti car can have the following combination of features:

1. Manual and Diesel

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Wrong</th>
<th>Cannot be answered from model</th>
<th>I don’t know</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>✅</td>
<td>❌</td>
<td></td>
<td>✗</td>
</tr>
</tbody>
</table>
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>
<tbody>
<tr>
<td></td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
A Skoda Yeti car can have the following combination of features:

3. Automatic and 4x4
A Skoda Yeti car can have the following combination of features:

4. Adventure and 2-wheel-drive
A Skoda Yeti car can have the following combination of features:

5. Active and Diesel and Automatic

- Correct
- Wrong
- Cannot be answered from model
- I don't know
A Skoda Yeti car can have the following combination of features:

6. Diesel and Automatic and 4x4
A Skoda Yeti car can have the following combination of features:

- 7. Active and Benzin and 4x4  
  Correct  Wrong  Cannot be answered from model  I don’t know
A Skoda Yeti car can have the following combination of features:

8. Adventure and Manual and 4x4
A Skoda Yeti car can have the following combination of features:

9. Active and Benzin and Manual and 2-wheel-drive

Correct  Wrong  Cannot be answered from model  I don’t know

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

10. Automatic and Adventure and Benzin and 2-wheel-drive
Part Extras

This part of the questionnaire refers to the following model (the model will be repeated on each page):
A Skoda Yeti car can have the following combination of features:

1. Parking-Heater and Styling-Package

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

2. Panorama-Roof and Offroad-Styling

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

3. Parking-Heater and Offroad-Styling
A Skoda Yeti car can have the following combination of features:

4. Parking-Heater and Heated-Front-Pane
A Skoda Yeti car can have the following combination of features:

5. Parking-Heater and Styling-Package and Offroad-Styling

<table>
<thead>
<tr>
<th></th>
<th>Correct</th>
<th>Wrong</th>
<th>Cannot be answered from model</th>
<th>I don't know</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parking-Heater</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glass1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heated Front Pane</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sunset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Panorama Roof</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Styling Package</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Glass3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heated Front Pane</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sunset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


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

6. Sunset and Parking-Heater and Styling-Package
A Skoda Yeti car can have the following combination of features:

- Heated-Front-Pane and Sunset and Panorama-Roof

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

8. Sunset and Panorama-Roof and Parking-Heater and Offroad-Styling

Correct  Wrong  Cannot be answered from model  I don’t know

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

9. Heated-Front-Pane and Sunset and Styling-Package and Offroad-Styling
A Skoda Yeti car can have the following combination of features:

10. Heated-Front-Pane and Sunset and Panorama-Roof and Styling-Package
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

PrinterPool

fax  printer

PrinterPool.fax implies PrinterPool.printer

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

```
PrinterPool

Printer [1..*]

color PrinterHead

color implies colorPrinterHead
```
Existential Quantification (Full OCL)

PrinterPool [0..*]  
Printer [1..*]  
colorCapable  

colorCapable = Printer->exists(color)
Univ. Quantification (Complete OCL)

allColor = Printer->forAll(color)
Use Context to Simplify Quantifiers

PrinterPool [0..*]

Printer [1..*]

fax

color

color

fax.color implies
Printer->forall(color)

Fax.color implies color

color

color

PrinterPool [0..*]

Printer [1..*]

Fax
Cardinality Constraints

PrinterPool [0..*]

Printer [1..*]

default

Printer->select(default)->size() <= 1
Arithmetic Constraint w/ VClassifier

PrinterPool [0..*]

Printer [1..*]
resolution : int

Scanner [0..*]
resolution : int

Printer->max(resolution) <= Scanner->max(resolution)

Printer [1..*]
resolution : int

Scanner [0..*]
resolution : int

Printer->max(resolution) <= Scanner->max(resolution)

Printer [1..*]
resolution : int

Scanner [0..*]
resolution : int

Printer->max(resolution) <= Scanner->max(resolution)

Printer [1..*]
resolution : int

Scanner [0..*]
resolution : int

Printer->max(resolution) <= Scanner->max(resolution)

Printer [1..*]
resolution : int

Scanner [0..*]
resolution : int

Printer->max(resolution) <= Scanner->max(resolution)
Task Description: Skoda Yeti Laurin & Klement

- Skoda has a top-of-the-range edition called Laurin and Klement named after the two founders of Skoda, namely Vaclav Laurin and Vaclav Klement.
- Our modeling task focuses on this top-of-the-range edition and on its diesel cars.
- These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).
- The Laurin and Klement range offers as default a lot of luxury features, but there are still some features which may be selected as extras. The customer can choose parking assistant, backing sensor, double trunk floor or extra wheel. However, choosing the parking assistant excludes choosing the backing sensor. In addition, if parking assistant is chosen, the car needs to have the strongest engine (140hp).
These cars come with automatic as well as manual gearbox, but when it is automatic, only the 4x4 drive and a 140hp engine are possible. If the customer opts for a two-wheel drive s/he must choose the manual shift and a 110hp engine. The manual shift and the 4x4 drive give the alternatives of both engines (140hp or 110hp).
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
- Object Substitution
- Object Existence
  - BW
  - Color

Scanner

- Feeder speed: integer
- Turner

Quality
The VSpec layer of the Scanner

Scanner

speed: integer

BothSides

Type

1..1

Color

BW
One resolution of the Scanner

Scanner

- Type = True
- Speed = 200
- BothSides = False
- Color = True
- BW = False

Type

- BothSides = False

BothSides

- speed: integer

Scanner

- speed: integer
Realization Layer of the Scanner (Object)

ParametricSlotAssignment
slotIdentifier="speed"

Scanner

f: Feeder
speed=100

q: Quality

t: Turner

Feeder
speed: integer

Turner

Quality

Color

BW

ObjectSubstitution

ObjectSubstitution

Scanner

speed: integer

BothSides

Type

1..1

Color

BW
The resulting product of the Scanner

Scanner

f: Feeder
speed = 200

q: Color

Turner

Color

BW

Scanner = True

speed = 200

BothSides = False

Type = True

Color = True

BW = False
The base model of the PrinterPool

PrinterPool

p:Printer[1..*]

Printer

color:Boolean
speed:String

PX
The VSpec layer of the PrinterPool

Diagram:

```
PrinterPool
  ↓
Printer [*]
  ↓
  ↓
Color  Speed
  ↓  1..1
  ↓
X20  X22
```
One resolution of PrinterPool

PrinterPool

PrinterPool = True

VInst1: Printer

Speed = True

Color = True

X20 = True

X22 = True

VInst2: Printer

Speed = True

Color = False

X20 = False

X22 = True

1..1
Realization Layer of the PrinterPool

PrinterPool

\[ p : \text{Printer}[1..*] \]

Printer

- color: Boolean
- speed: String

SlotAssignment

- slotIdentifier = "speed", value = "X20"
- slotIdentifier = "speed", value = "X22"

FragmentSubstitution

multi = True

PrinterPool

Color

Speed

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 pPrinter.

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 showing relationships between `PrinterPig`, `Printer`, `MainPower`, `Adapter`, `EmergencyPower`, `acmePower`, and `TexasPower`. The `MainPower` class has attributes `defaultAttribute` and `powerType`. The diagram illustrates how these classes interact within the context of a printer component.
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

Opaque Variation Point 1

Opaque Variation Point 2

Source Object

Place Holder

Base Models

Semantic Spec

Special Substitution

Printer

Main Power Supply

Acme Power

Texas Power

Emergency Power Supply

Max Number of Adapters: Int

<1..1>
Recall CVL Architecture

- Configurable Units
- Variability Interfaces
- Variation Points
- Variability Realization
- VSeps
- Constraints
- Variability Abstraction
- Resolutions
- 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
Configurable Unit: How?

- Technically a new kind of variation point
- Entire CU is a single variation point
- References an entire object of base model, which becomes a variability container
- CU contains other variation points
  - inner variability
  - Composite Variation Point
Configurable Unit and VInterface

- **Power**
  - Acme
  - Texas

- **ObjectExistence**
  - PrinterPackage

- **PrinterPackage**
  - AcmePower
  - TexasPower

- **CU**
- **VInterface**
- **CVSpec**

- **Pr_Cv**

- **Binding**
- **Base model reference**
Composition of Configurable Units
Resolution of Complex CU

To materialize Office_CU, provide a VConfiguration that resolves the CVSpec Office_Cv.
Reusing Variability Interfaces

Though sharing the same interface, the laser and inkjet printers are configured separately (since the VSpecs are separate).
Resolution in Reuse Scenario
CVL – Early Experiments

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

- Elevator
- Paper Cutting Machine
- Pumps
Safe Power Drive System – PDS (SR)

- Plug-in safety module
- Safety Module
- Motor Controller
- STO (Safe Torque Off - cut power to motor)
- Optional tachometer (Speed and direction)
- Optional brake
- Digital I/O
- Com
- Power
- Drive

Optional brake

STO (Safe Torque Off - cut power to motor)
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, 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 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)

Executable Testing Code

Product Configuration File
Variability and domain design

Derivation

Deployment
The Executable Base Model: UML on Papyrus with profiles

Composite structure(s)

«SUT» Safety Module

«Test Component» Motor Controller

«Test Component» Motor

State Machine(s)

Variability and domain design

Derivation

Deployment

UML Fragments

Base UML Model incl. tests
Defining the Product Line with CVL Tool and Papyrus

CVL Abstraction Layer

CVL Realization Layer

Coloring the effects of the substitution
The Dual Product Line

- Some features are defined at design time while others are defined at runtime
- We define a dual product line with
  - One abstraction layer
  - Two realization layers
    - One for the design time variability
    - One for the corresponding runtime options
Defining the restricted configuration file

Selection = conveyor belt

Configuration template only containing the parameters to set the safety functions of any conveyor belt

Ecore (meta)model

Configuration file to setup a given instance of the conveyor belt

Conform model instantiated using Ecore/EMF built-in mechanisms
Variants of behavior: changes to a state machine

- **Substitution**
- **Replacement**
- **Placement**
Execute CVL to obtain set of product models

- Execute CVL
  - on design time model
  - on configuration file model
UTP Test Cases integrated with model

Test component port

Test component port

Test component part

SUT

Test component part

SUT

Variability and domain design

Derivation

Deployment

10/4/2013
CVL – Common Variability Language
81
Testing a product line

Variability and domain design

- Combinatorial Coverage arrays
  - select
  - CVL metamodel

The Product Line: CVL Abstraction Layer

- CVL Resolution

CVL Realization Layer

The Product model with tests

- generate

Derivation

- generate

Deployment

- compile and run
- execute tests

Product Simulation Code

Executable JUnit Testing Code

Base DSL Model incl. tests

DSL Fragments

CVL metamodel (+ profiles)

10/4/2013 CVL – Common Variability Language