What is needed to handle Variability?

• Define **mandatory** properties and functionalities

• Define **optional** choices: 0, 1..N, choice among n

• **Variants**

• **Constraints**
  – Dependency
  – Mutual exclusion
How to express variability?

- Inheritance *(design, implementation)*
- Genericity *(design, implementation)*
- Design Pattern *(design, implementation)*
- Aspect-Oriented Design & Programming *(design, implementation)*
- Model Transformation *(design)*
- Model Composition *(design)*

  - **Feature Models** *(requirement engineering, design, also implementation)*
    - Similar technique: **Decision Modeling**
Variability Model
Feature Model: \textit{de facto} standard

• Research
  – 2500+ citations of [Kang et al., 1990] on Google Scholar
  – Central to many generative approaches
    • at requirements or code level
  – Tools & Languages (GUIDSL/FeatureIDE, SPLOT, FaMa, etc.)

• Industry
  – Tools (Gears, pure::variants),
  – Will be Part of Common Variability Language (CVL), future OMG standard
R8 Spyder
5.2 FSI quattro R tronic

Prix total
171.216,00 EUR

Prix de base
170.490,00 EUR

Equipements optionnels
726,00 EUR

- Informations détaillées
- Entrez l'Audi Code
- Générer un PDF
- Nouvelle configuration

Packs
Aucun pack n'est proposé pour ce modèle.

Couleurs
Blanc Ibis
Noir
Prix: 0,00 EUR

Couleurs métallisées à partir de 0,00 EUR

Couleurs à effet perlé à partir de 0,00 EUR

Couleurs personnalisées Audi exclusive

Jantes
4 Jantes alu 5 BRANCHES ROTOR finition titane 8,5 x 19 à l'avant, 11 x 19 à l'arrière. Pneus 235/35 R19 à l'avant et 305 /30 R19 à l'arrière
Prix: 726,00 EUR

19" à partir de 0,00 EUR
Excludes

Régulateur de vitesse

Système d'aide au stationnement APS avant / arrière

Système d'aide au stationnement APS avant / arrière avec affichage dans l'écran MMI

Système d'aide au stationnement Advanced : APS avant et arrière et caméra arrière

Audi hill assist : assistance au démarrage en côte

Réinitialiser la sélection

Attention:
Le prix peut varier en fonction du choix de moteur et des équipements.

Un aperçu des équipements:

Mode expert

1. Modèle
2. Moteur
3. Extérieur
4. Intérieur
5. Option
6. Votre Audi
Vérification de votre sélection

Cet équipement nécessite un équipement complémentaire:

- ✔ GPS Plus avec disque dur

Voici les équipements complémentaires possibles:

- ⚪ Ordinateur de bord en couleur avec programme efficiency

  Remarque: uniquement sur les modèles avec système Start-Stop et uniquement disponible en combinaison avec l'autoradio Concert, l'autoradio Symphony ou un système de navigation

- ⚪ Pack Intenso Plus

  Sans appareil de navigation

Série

2.934,25 EUR

Attention:
Le prix peut varier en fonction du choix de moteur et des équipements.

Un aperç des équipements:

Mode expert

Réinitialiser la sélection
Feature Models

- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights → FrontFogLights
Hierarchy: rooted tree

Variability:
- mandatory,
- optional,
- Groups: exclusive or inclusive features
- Cross-tree constraints
Hierarchy + Variability
= set of valid configurations

configuration = set of features selected

\{CarEquipment, Comfort, DrivingAndSafety, Healing, AirConditioning, FrontFogLights\}
Hierarchy + Variability
= set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, Air Conditioning}
Hierarchy + Variability

= set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning, AirConditioningFrontAndRear, FrontFogLights}
Hierarchy + Variability
=
set of valid configurations

configuration = set of features selected

{CarEquipment, Comfort, DrivingAndSafety, Healthing, AirConditioning, AutomaticHeadLights}
Hierarchy + Variability

= set of valid configurations

{CarEquipment, Comfort, DrivingAndSafety, Healing}

{AirConditioning, FrontFogLights}
{AutomaticHeadLights, AirConditioning, FrontFogLights}
{AutomaticHeadLights, FrontFogLights, AirConditioningFrontAndRear}
{AirConditioningFrontAndRear}
{AirConditioning}
{AirConditioningFrontAndRear, FrontFogLights}
Feature Models

Metamodel

Model

Modeling Language

System Under Study

\( \chi \) : Conformant To
\( \mu \) : Representation Of
\( \varepsilon \) : Element Of

- **CarEquipment**
  - **Healthing**
  - **AirConditioningFrontAndRear**
  - **AirConditioning**
  - **Comfort**
    - **AutomaticHeadLights**
  - **DrivingAndSafety**
    - **FrontFogLights**

- **Constraints**
  - **AutomaticHeadLights** \( \Rightarrow \) **FrontFogLights**
Managing variability models with FAMILIAR
#1 Automated Analysis

- CarEquipment
  - Healthing
    - AirConditioningFrontAndRear
    - AirConditioning
  - Comfort
    - AutomaticHeadLights
  - DrivingAndSafety
    - FrontFogLights
- Constraints
  - AutomaticHeadLights → FrontFogLights
#2 Multiple Feature Models

```
▲ CarEquipment
  ▼● Healthing
  ▼▲ AirConditioningFrontAndRear
  ▲ AirConditioning
  ▼● Comfort
    ▲ AutomaticHeadLights
    ∂ FrontFogLights
  ▼● DrivingAndSafety
  ▼Constraints
    AutomaticHeadLights → FrontFogLights

▲ CarEquipment
  ▼● Healthing
  ▲ AirConditioningFrontAndRear
  ▲ AirConditioning
  ▼● Comfort
    ▲ AutomaticHeadLights
    ∂ FrontFogLights
  ▼● DrivingAndSafety
  ▼Constraints
    AutomaticHeadLights → FrontFogLights

```

Two Key Requirements

• **#1 Automated analysis**
  – Aka support to better understand and play with your feature model (TVL model)

  "Automated analysis of feature models 20 years later: A literature review"
  David Benavides *, Sergio Segura, Antonio Ruiz-Cortés

• **#2 Managing multiple feature models**
  – Composing / Decomposing / Diff and Reasoning about their relationships
  – Combining these operators
FAMILIAR language and environment

// foo.fml
fm1 = FM ("foo1.tvl")
fm2 = FM ("foo2.m")
fm3 = merge intersection { fm1 fm2 }
c3 = counting fm3
renameFeature fm3.TV as "OutputTV"
fm5 = aggregate { fm3 FM ("foo4.xml") }
assert (isValid fm5)
fm6 = slice fm5 including fm5.TV.*
export fm6

Interoperability  Language facilities  Environment
FAMILIAR ... features

Interoperability

- fm1 = FM("foo.tvl")
- fm2 = FM("foo.m")
- fm3 = FM("foo.xmi")
- fm4 = FM(A : B ....)

Serialization

- serialize fm4 into SPLOT
- serialize fm1 into featureide

Confidentiality

- config
- cores
- deads
- falseOptionals
- cleanup
- configuration

De/Composition

- merge
- diff
- intersection
- union
- insert
- aggregate
- map
- unmap
- slicing
- extract
- unmap

Reasoning

- isValid
- compare
- counting
- configs
- select
- deselect
- asFM
- merge
- diff
- intersection
- union
- insert
- aggregate
- map
- unmap
- slicing

Editing

- renameFeature
- removeFeature
- setOptional
- setMandatory
- setAlternatives
- setOr
- accessor
- copy
- extract
- unmap

Language Facilities

- fm1.*
- fm1.B
- iterator/conditional
- assertion
- modular mechanisms
- restricted set of types

Language Facilities
Hello World

```
fml1 = FM ("foo1.tv1")
s = "hello world"
c = counting fml
n = size fml.*
println s
```
Typed language

- Domain-specific types
  - Feature Model,
  - Configuration,
  - Feature,
  - Constraint

- Other types include
  - Set
  - String
  - Boolean,
  - Enum,
  - Integer and Real.

- A set of operations, called operators, are defined for a given type.

```fml
FML = "foo1.tvl"
f1 = root FML
f2 = FML.B
fts = FML.*
n = size fts
n2 = cores FML

cf = configuration FML
select B in cf
deselect C in cf

cst1 = constraint (B -> !C)
addConstraint cst1 to FML
```
Typed language

```plaintext
fml = FM ("fool.tvl")
ft1 = root fml
ft2 = fml.B
fts = fml.*
n = size fts
n2 = cores fml

cf = configuration fml
select B in cf
deselect C in cf

cst1 = constraint (B -> !C)
addConstraint cst1 to fml
```
fm1 = FM("fool.tvl")
ft1 = root fm1
ft2 = fm1.B
fts = fm1.*
n = size fts
n2 = cores fm1

cf = configuration fm1
select B in cf
deselect C in cf

cst1 = constraint (B -> !C)
addConstraint cst1 to fm1
Importing/Exporting feature models

Internal notation or by "filename extensions"

```plaintext
fm1 = FM ("fool.tvl")
fm2 = FM (A : [B] [C] D ; )
fm3 = FM ("foo2.m")
serialize fm2 into SPLLOT  // export
```
Feature Accessors (1)

\[ \text{fm1} = \text{FM} (A : B [C] ; B : E F ; C : (I | J) ; ) \]

\[ \text{r1} = \text{root} \ \text{fm1} \]
\[ \text{s} = \text{children} \ \text{r1} \]
\[ \text{s1} = \text{children} \ \text{fm1}.A \]
\[ \text{assert} \ (\text{s} \ \text{eq} \ \text{s1}) \ // \text{equality of the two sets} \]
\[ \text{ft1} = \text{parent} \ \text{fm1}.F \]
\[ \text{str1} = \text{name} \ \text{ft1} \]
\[ \text{ft2} = \text{parent} \ F \ // \text{parent fm1}.F \]

// another FM
\[ \text{fm2} = \text{FM} (A : B C E ; C : (I | J)+ ; ) \]
\[ \text{ft3} = \text{fm2}.B \]
\[ \text{ft4} = \text{name} \ B \ // \text{ambiguity} \]
Other constructs

\[
\text{fml} = \text{FM} \ (A: B \ [C] \ D; \ D : (E | F) +; \ F : (I | J | K); \ E : [Z];)
\]

\[
fml\text{bis} = \text{copy fml} \ // \text{save the original version}
\]

\[
\text{renameFeature fml.B as "Bbis"}
\]

\[
\text{sl} = \text{fml.*} \ // \text{set of features of fml}
\]

\[
\text{foreach (ftl in sl) do}
\]

\[
\text{println ftl}
\]

\[
\text{end}
\]
Configuration

\[ \text{conf.fml} \]

```
fm1 = FM (A: B [C] [D] E; D: (F|G); E: (I|J|K|L); C -> !D;

cl = configuration fm1
select C in cl
scl = selectedF cl // accessors
CFM1 = asFM cl // configuration and FM: back!
```

FAMILIAR (for FeAture Model scrIpt Language for manIplementation and Automatic Re
University of Nice Sophia Antipolis, UMR CNRS 6070, I3S Laboratory
https://nyx.unice.fr/projects/familiar/

```
fm1> cFM1
CFM1: (FEATURE_MODEL) A: B E C;
E: (J|L|I|K);
A;
B;
C;
fm1> fm1
fm1: (FEATURE_MODEL) A: [D] B E [C];
D: (F|G);
E: (J|L|I|K);
(C -> !D);
```
Operations for Feature Models (1)

\[ \text{fm1} = \text{FM} \left( \text{A : [B] [C] ; B \rightarrow !C ; B \textbf{and} C ; } \right) \]

\[ \text{b1} = \text{isValid} \ \text{fm1} \]

FAMILIAR (for FeAture Model scrIpt Language for manIpulation and Automatic Reasoning)  
University of Nice Sophia Antipolis, UMR CNRS 6070, I3S Laboratory  
https://nyx.unice.fr/projects/familiar/

```
fml> ls  
(FEATURE_MODEL) fm1  
(BOOLEAN) b1  
fm1> b1  
b1: (BOOLEAN) false  
fm1> configs fm1  
res0: (SET) {}  
```

operatorsFM.fml
Operations for Feature Models (2)

```plaintext
1. fm1 = FM (W: P (T|U); P: (R|S)+ ; T: [V] [A]; R -> !V; S -> U; R -> A ;)
2. b1 = isValid fm1
3. s1 = configs fm1
4. c1 = counting fm1
5. dfm1 = deads fm1
6. println "cores: ", cores fm1
7. fo1 = falseOptionals fm1
```

```
achcr-scr:FML-scripts macher$ java -jar -Xmx1024m operatorsFM2.fml
cores: {P;W}
FAMILIAR (for FeAture Model scrIpt Language for University of Nice Sophia Antipolis, UMR CNRS 6072, https://nyx.unice.fr/projects/familiar/
fm1> ls
(SET) fo1
(SET) dfm1
(SET) s1
(DOUBLE) c1
(BOOLEAN) b1
(FEATURE_MODEL) fm1
fm1> c1
  c1: (DOUBLE) 2.0
fm1> fo1
  fo1: (SET) {A}
fm1> dfm1
  dfm1: (SET) {V}
```
Operations for Feature Models (3)

\[
fml = \textbf{FM} (A : [B] [C] ; B \rightarrow !C ; B \textbf{ and } C ; )
\]
\[
b1 = \textbf{isValid} \ fml
\]
\[
cstsl = \textbf{constraints} \ fml
\]
\[
\textbf{foreach} (\text{cst in cstsl}) \textbf{ do}
\]
\[
\text{println} \ "\text{removing constraint... }" , \text{cst}
\]
\[
\text{removeConstraint} \ \text{cst in fml}
\]
\[
c = \textbf{counting} \ fml
\]
\[
\text{println} \ "\text{now the number of valid configurations is... }" , \text{c}
\]

```bash
MacBook-Pro-de-Mathieu-2:FML-scripts macker$ java -jar -Xmx1024M ../FML-0.9.9.5.jar operatorsFM3.fml
removing constraint... (B & C)
now the number of valid configurations is... 3.0
removing constraint... (B \rightarrow !C)
now the number of valid configurations is... 4.0
```
Multiple Feature Models

SPL/internal/software variability
(Pohl et al. 2005, Metzger 2007)

PL/external variability
(Pohl et al. 2005, Metzger 2007)

context variability
(FORM 1998, Tun et al. 2009 (problem world), Hartmann 2008 (CVM), Lee et al. 2010)

Concern 1, 2, 3, ..., n
View 1, 2, 3, ..., n
(Dunghana et al. 2010, Hubaux et al. 2010, Zaid et al. 2010)

Stakeholder 1, 2, 3, ..., n
SoC support = Composition/Decomposition for managing large, complex and multiple feature models

Composing Feature Models (1)

fm1 = \textbf{FM} (A : B [C] [D] ; D : (E|F) ; C \rightarrow \neg E; )
fm2 = \textbf{FM} (I : J [K] L ; )
fm3 = \textbf{FM} (M : (N|O|P)+ ; )
cst = \textbf{constraints} (J \textbf{ implies } N ; )

// equivalent to aggregate \{ fm1 fm2 fm3 \}
fm4 = \textbf{aggregate} fm* \textbf{ withMapping} cst
Composing Feature Models (2)

\[
\begin{align*}
\text{fm1} &= \text{FM} (A : B [C] [D] ; D : (E|F) ; C \rightarrow !E; ) \\
\text{fm2} &= \text{FM} (I : J [K] L ; ) \\
\text{fm3} &= \text{FM} (M : (N|O|P)^+ ; ) \\
\text{cst} &= \text{constraints} (J \text{ implies } C ; ) \\
// \text{ equivalent to aggregate \{ fm1 fm2 fm3 \} }
\text{fm4} &= \text{aggregate} \text{ fm* withMapping} \text{ cst}
\\
// \text{composition sometimes leads to "anomalies"}
\text{dfm4} &= \text{deads fm4}
\end{align*}
\]
Merging Feature Models

mergeMI.fml

\[ \text{mergeMI.fml} \]

\[
\begin{align*}
\text{fmsupp1} &= \text{FM} \text{ (MedicalImage : [Anonymized] MRI [DICOM]; MRI : (T1|T2));} \\
\text{fmsupp2} &= \text{FM} \text{ (MedicalImage : Anonymized MRI [Header]; MRI : [T1] [T2]);} \\
\text{fmsupp3} &= \text{FM} \text{ (MedicalImage : [Anonymized] MRI [DICOM]; MRI : [T1] [T2]; T1 \rightarrow Anonymized;)} \\
\end{align*}
\]

\[
\begin{align*}
\text{s1} &= \text{configs fmsupp1} \\
\text{s2} &= \text{configs fmsupp2} \\
\text{s3} &= \text{configs fmsupp3} \\
\text{s123} &= \text{setUnion s3 setUnion s1 s2} \\
\text{fmSupp} &= \text{merge union fmsupp*} \\
\text{assert (size s123 eq counting fmSupp)} \\
\text{fmCommon} &= \text{merge intersection \{} fmsupp1 fmsupp2 \text{ \}} \\
\text{sC} &= \text{configs fmCommon} \\
\text{sC2} &= \text{setIntersection s1 s2}
\end{align*}
\]
Comparing Feature Models

\[ fm0 = \text{FM} \left( A : (B \mid C) \right); \]
\[ fm1 = \text{FM} \left( A : (B \mid C)^+ \right); \]

assert \( (\text{cmp10 \ eq \ GENERALIZATION}) \)

// example taken from Automated Analysis of Feature Models

\[ fm2 = \text{FM} \left( A : B \ [C] \ [D] \right); \]
\[ fm3 = \text{FM} \left( A : B \ [C]; \ B : [D] \right); \]
\[ \text{cmp23} = \text{compare} \ fm2 \ fm3 \]

see also Thuem, Kastner and Batory, ICSE’09
Putting all together: Example 1
Merge Intersection: Available Suppliers

Suppliers? Products?

A customer has some requirements
In FAMILIAR

REGsupp1 = FM (Registration : Header Format Modality [Affine] ;
                Header : (v1|v3);
                Format : (DICOM|Nifti);
                Modality : CT; )

REGsupp2 = FM (Registration : Header [Affine] Format Modality ;
                Header : (v1|v2);

REGsupp3p: (FEATURE_MODEL) False

REGsupp1p: (FEATURE_MODEL) Registration: Header Modality Format ;
          Header: v1 ;
          Modality: CT ;
          Format: (DICOM|Nifti);

REGsupp1p = merge intersection { REGrequired REGsupp1p }
Merge Union: Availability Checking

Yes!
Can suppliers provide all products?

“compare”
In FAMILIAR

REGsuppl = **FM** (Registration : Header Format Modality [Affine] ;

Header : (v1|v3);
Format : (DICOM|Nifti) ;
Modality : CT; )

REGsupp2 = **FM** (Registration : Header [Affine] Format Modality ;

Header : (v1|v2);
Format : (Nifti|GE) ;
Modality : MRI; )

REGsupp3 = **FM** (Registration : Header [Rigid] Format Modality ;

Header : (v2|v3) ;
Format : Nifti ;
Modality : (MRI|PET); )

REGrequired = **FM** (Registration : Header Format Modality ;

Header : v1 ; //v3;
Format : (DICOM|Nifti) ;
Modality: (MRI|CT);
!DICOM or !MRI;
)

REGmspl = **merge sunion** REGsupp*     // merge all FMs whose variable identifier starts w.

cmp = **compare** REGrequired REGmspl
//missingSPL = merge diff { REGrequired REGmspl }

82
Merging operation: implementation issues

Merging onoperation:

implementaAon

issues

Merging operation: implementation issues

Merging onoperation:

implementaAon

issues

How to synthesise a feature model that represents the union of input sets of configurations?

```
fmsupp1 = FM (MedicalImage : [Anonymized] MRI [DICOM] ; MRI : (T1|T2) ; )
fmsupp2 = FM (MedicalImage : Anonymized MRI [Header] ; MRI : [T1] [T2] ; )
fmsupp3 = FM (MedicalImage : [Anonymized] MRI [DICOM] ; MRI : [T1] [T2] ; T1 -> Anonymized; )

// computing the union of sets of configurations like this is COSTLY
s1 = configs fmsupp1
s2 = configs fmsupp2
s3 = configs fmsupp3
s123 = setUnion s3 setUnion s1 s2

// you WONT scale
//...
fmSupp = merge subunion fmsupp*

assert (size s123 eq counting fmSupp)
```
Merging operation: semantic issues (2)

\[
\varphi
\]

\[
\begin{align*}
\{M I, M A, F, C T, N i f t i\}, \\
\{M I, M A, F, C T, N i f t i, A N\}, \\
\{M I, M A, F, D I C O M, M R I, A N\}
\end{align*}
\]

Union

Intersection

Diff

How to synthesise a feature model that represents the union of input sets of configurations?

Fig. 2: For a given set of configurations, three possible yet different FMs \((s_0 = [f_{m_0}] = [f_{m_1}] = [f_{m_2}])\)
Merging operation: algorithm

\[ \varphi_1 \]
\[ \varphi_2 \]
\[ \varphi_3 \]

How to synthesise a feature model that represents the union of input sets of configurations?

\[ \varphi_{123} \]

merged propositional formula

merged hierarchy

Set mandatory features
Detect Xor and Or-groups
Compute “implies/excludes” constraints

see also [Czarnecki SPLC’07 or SPLC’12]
Building “views” of a feature model
Building “views” of a feature model

• Problem: given a feature model, how to decompose it into smaller feature models?

• Semantics?
  – What’s the hierarchy
  – What’s the set of configurations?
A first try

Problem: You can select A3 without A5

Hierarchy and Configuration matter!
**Slicing Operator**

**slicing criterion**: an arbitrary set of features, relevant for a feature model user

**slice**: a new feature model, representing a projected set of configurations
Slicing operator: going into details projected set of configurations

\[ \text{fm1p} = \{ \{D,E,T\}, \{S,T\}, \{B,E,T\}, \{S,T\}, \{S,T\}, \{S,T\}, \{D,E,T\} \} \]


**Constraints**
- E implies D
- R implies E
- D excludes F
- S implies (F andnot E)
Slicing operator: going into details synthesizing the corresponding feature model

\[ \varphi_1 \Downarrow \varphi_{s1} \]

existential quantification of features not included in the slicing criterion

\[
f_{m1} \]

constraints
E implies D
R implies E
D excludes F
S implies (F andnot E)

\[
f_{m1p} = \{ \{D,E,T\}, \{S,T\} \} \]

see also [Acher et al., ASE’11/AOSD’12]
Slicing operator with FAMILIAR (1)

\[ fm1 = \text{FM} \ (W : P \ T \ [U] ; \ T : [V] \ A ; \ A : B \ C \ [D] ; \ C : [E] \ [F] ; \ P : (R|S)+ ; \ E \ implies \ D ; \ R \ implies \ E ; \ S \ implies (F \ and \ !E) ; \ D \ implies \ !F ; \) \]

\[ fm2 = \text{slice} \ fm1 \ including \ \{ \ S \ T \ E \ D \ \} \]
\[ fm2bis = \text{slice} \ fm1 \ excluding \ \{ \ W \ P \ R \ V \ A \ B \ C \ F \ U \ \} \]

\[ \text{cmp} = \text{compare} \ fm2 \ fm2bis \]
\[ \text{assert} \ (\text{cmp eq \ REFACTORING}) \]
Slicing with FAMILIAR (2)

\[
\begin{align*}
\text{fm1} &= \text{FM} (W : P T [U] ; T : [V] A ; \\
&\hspace{1cm} A : B C [D] ; \\
&\hspace{2cm} C : [E] [F] ; \\
&\hspace{3cm} P : (R|S)+ ; \\
&\hspace{4cm} E \text{ implies } D ; R \text{ implies } E ; \\
&\hspace{5cm} S \text{ implies } (F \text{ and } !E) ; D \text{ implies } !F ; )
\end{align*}
\]

\[
\begin{align*}
\text{fm2} &= \text{slice } \text{fm1 including fm1.A}.* ++ \{ \text{fm1.A} \} \\
\text{fm3} &= \text{slice } \text{fm1 including fm1.P}.* ++ \{ \text{fm1.P} \} \\
&\hspace{1cm} // \text{fm3bis} = \text{slice } \text{fm1 including } \{ \text{fm1.P, fm1.R, fm1.S} \} // \text{equivalent to fm3} \\
\text{fm4} &= \text{slice } \text{fm1 including } \{ \text{fm1.E, fm1.D, fm1.F} \} \\
\text{fts5} &= \{ \text{fm1.P, fm1.W} \} ++ \text{fm1.P}.* \\
\text{fm5} &= \text{slice } \text{fm1 including } \text{fts5}
\end{align*}
\]
Putting all together: Example 2
From marketing, customers, product management

From existing software assets (technical variability)

From marketing, customers, product management

[V1] - [V2] - [V3]

[V1] ↔ f1
[V2] ↔ f2
[V3] ↔ f3

usefulness
realizability

From existing software assets
Realizability checking

1. aggregate

2. slice ("realizable part")

3. compare

4. merge diff ("unrealizable products")

see also [Acher et al. AOSD’12 and CAiSE’12]
With FAMILIAR

FMSoftware = FM (R : (F1|F2) F3 [F4] ; )

* Figure 1, Section 3

* Metzger et al. 2007, RE'07

With FAMILIAR

https://nyx.unice.fr/projects/familiar/

realizibility.fml
Putting all together: Example 3
#1 Reverse Engineering Architectural Feature Models

Case Study: FraSCAti Architecture

Collaboration with Anthony Cleve (University of Namur / PRECISE, Belgium), Philippe Collet and Philippe Lahire (University of Nice Sophia Antipolis), Philippe Merle and Laurence Duchien (University of Lille / INRIA)

[Acher et al., ECSA’11]
[Acher et al., BENEVOL’11]
[Acher et al., GDR GPL’12]
Extraction process

Software Artefacts

Variability Modeling

Software Architect View

Automatic Extraction

Philippe Merle, software architect of FraSCAti

Combination of plugin dependencies and hierarchical component model to synthesise a feature model
**Highlights**

• **Automated Procedure**
  – Extracting and **Combining** Variability Sources (incl. software architect knowledge)
  – Advanced feature modeling techniques have been developed (tool supported with FAMILIAR)

• **Some Lessons Learned**
  – Extraction procedure yields promising results
  – Essential role of software architect
    • To validate the extracted feature model
    • To integrate knowledge

• **Extensions**
  – Evolution of FraSCAti with DIFF (v1.3, v1.4, etc.)
Collaboration with Patrick Heymans, Anthony Cleve, Gilles Perrouin (University of Namur / PRECISE, Belgium), Philippe Collet and Philippe Lahire (University of Nice Sophia Antipolis),

[Acher et al., VaMoS'12]
**Manual** extraction of a feature model from product description(s) is not possible

<table>
<thead>
<tr>
<th>Identifier</th>
<th>License</th>
<th>Language</th>
<th>Storage</th>
<th>LicenseCostFee</th>
<th>RSS</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confluence</td>
<td>Commercial</td>
<td>Java</td>
<td>Database</td>
<td>US10</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PBwiki</td>
<td>NoLimit</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MoinMoin</td>
<td>GPL</td>
<td>Python</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DokuWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PmWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DrupalWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Database</td>
<td>Different Licences</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TWiki</td>
<td>GPL</td>
<td>Perl</td>
<td>FileRCS</td>
<td>Community</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>GPL</td>
<td>PHP</td>
<td>Database</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Exact set of configurations, each configuration corresponding to at least one product

640 configurations (634 counter examples)
Automation

- Each product description is encoded as a feature model

<table>
<thead>
<tr>
<th>Identifier</th>
<th>License</th>
<th>Language</th>
<th>Storage</th>
<th>LicenseCostFee</th>
<th>RSS</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confluence</td>
<td>Commercial</td>
<td>Java</td>
<td>Database</td>
<td>US10</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PBwiki</td>
<td>Nolimit</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MoinMoin</td>
<td>GPL</td>
<td>Python</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DokuWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PmWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Files</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DrupalWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Database</td>
<td>Different Licences</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TWiki</td>
<td>GPL</td>
<td>Perl</td>
<td>FilesRCS</td>
<td>Community</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MediaWiki</td>
<td>GPL</td>
<td>PHP</td>
<td>Database</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- Feature models \{fm1, fm2, ..., fm8\} are merged
  - Output: a new feature model
    - Configuration: union of input sets of configurations
    - Hierarchy: by default, we exploit the structure of the tabular data
      - Can be overridden by specific user directives
  - VariCell
    - DSL built on top of FAMILIAR
Putting all together: Example 4
Modeling Variability From Requirements to Runtime

The case of video surveillance processing chains

Adaptive systems

Collaboration with Sabine Moisan and Jean-Paul Rigault (INRIA)
Video surveillance processing chains

large number of software configurations for a large number of requirements
Video surveillance processing chains

Software Product Line (SPL) approach

Video Surveillance Application Requirements

Software Platform Configurations
Implementation: under the hood

Video surveillance application

- Task
  - Counting
  - Intrusion
- QoS
  - Quality
  - Object of interest
- Scene context
  - Scene description
  - Environment
  - Lighting
  - Camera
  - View
  - Artificial
  - Indoors
- Precision
- Response time
- Frame rate

Video surveillance platform

- Acquisition
- Segmentation
- Classification Clustering
- Model
  - Ellipse
  - Gravity
- Kernel function
- Traverse algorithm
  - Grid step
  - With window
  - Edge
  - Region
  - Color
  - Grey
Putting all together: Example 5 & Demo...
Realizing a Car Crash Management System SPL

bCMS – Requirements Definition Document

REQUIREMENTS DEFINITION DOCUMENT
FOR A SOFTWARE PRODUCT LINE OF CAR CRASH MANAGEMENT SYSTEMS
May 5, 2012
http://cserv0.site.uottawa.ca/cma2012

Authors:
Afredo Capozucca, Betty H.C. Cheng, Geri Georg, Nicolas Guelfi, Paul Istoan, Gunter Mussbacher

The authors would like to thank all participants of the 2011 AOM Bellairs Workshop for their contributions to this document: Omar Alam, Shaukat Ali, Robert France, Adam Jensen, Jean-Marc Jézéquel, Jörg Kienzle, Jacques Klein, Somayeh Malakuti, Sai Pradeep Mandalaparty, and Ana Moreira.
Scenario

• Construction of a domain feature model for the bCMS
  – Aggregation of functional and non functional parts
• Reasoning on the resulting bCMS FM
• Relating it to a platform feature model (J2EE from SPLOT repository)
• Checking realizability
  – While refining both feature models (bCMS / J2EE)
Summary: Variability Model Management

- CarEquipment
  - Heating
    - AirConditioningFrontAndRear
    - FrontFogLights
  - Comfort
    - AutomaticHeadLights
    - DrivingAndSafety
    - FrontFogLights
  - DrivingAndSafety
  - FrontFogLights
- Constraints
  - AutomaticHeadLights → FrontFogLights

FAMILIAR
Key Insights

[MOTIVATION/PROBLEM] Why modeling and managing Variability does and will matter

[SOLUTION FOR MANAGING FEATURE MODELS] Managing Variability Models with FAMILIAR

[APPLICATION FOR MODEL-BASED SPL ENGINEERING] Model-based variability engineering: applications, advanced topics, support
(ongoing) Comprehensive model-based product line support

Reverse engineering
Automated Analysis
Languages, API/DSLs

Evaluation (European and french projects, long-term collaboration with Thales, open source systems)
lightweight variability capture & understanding

Coupling with reverse engineering techniques

Beyond multiple viewpoints

Languages, API/DSLs

The Scala way...

Evaluation, empirical studies (French and European projects, collaboration with IT companies)