Putting all together: Example 1
Merge Intersection: Available Suppliers

A customer has some requirements

Suppliers? Products?
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);

---

REGsupp1p = merge intersection { REGrequired REGsupp1 }
REGsupp2p = merge intersection { REGrequired REGsupp2 }
REGsupp3p = merge intersection { REGrequired REGsupp3 }
Can suppliers provide all products?

"compare"
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);
            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 }
Merging operation: implementation 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 sunion fmsupp*

assert (size s123 eq counting fmSupp)
```

Anonymized v Header v DICOM v ~T1 v ~T2
Merging operation: semantic issues (2)

\[ s_0 = \{ \\
\{MI, MA, F, CT, Nifti\}, \\
\{MI, MA, F, CT, Nifti, AN\}, \\
\{MI, MA, F, DICOM, MRI, AN\} \\
\} \]

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 \( s_0 = [fm_0] = [fm_1] = [fm_2] \)
Merging operation: algorithm

\[ \phi_1 \quad \phi_2 \quad \phi_3 \]

\[ \phi_{123} \]

merged propositional formula

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

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

```
constraints
E implies D
R implies E
D excludes F
S implies (F and not E)
```

```
constraints
E implies D
D implies E
```
Slicing operator: going into details projected set of configurations

```
fm1 = {{A,B,C,D,E,P,R,T,U,W},
       {A,B,C,D,E,P,R,T,W},
       {A,B,C,F,P,S,T,V,W},
       {A,B,C,F,P,S,T,W},
       {A,B,C,D,E,P,R,T,V,W}}

fm1p = {{D,E,T},
         {S,T},
         {B,E,T},
         {S,T},
         {S,T},
         {D,E,T}}
```
Slicing operator: going into details
synthesizing the corresponding feature model

 existential quantification of features not included in the slicing criterion

\( \varphi_1 \)

\( \varphi_{s1} \)

\( \text{fm1p} = \{ \{D,E,T\}, \{S,T\} \} \)

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

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

\[
\begin{align*}
\text{fm2} &= \text{slice fml including } \{ S T E D \} \\
\text{fm2bis} &= \text{slice fml excluding } \{ W P R V A B C F U \}
\end{align*}
\]

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

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

\[ fml2 = \text{slice} \ fml \ \text{including} \ fml.A.* \ ++ \ \{ \ fml.A \} \]

\[ fml3 = \text{slice} \ fml \ \text{including} \ fml.P.* \ ++ \ \{ \ fml.P \} \]
\[ // fml3bis = \text{slice} \ fml \ \text{including} \ \{ \ fml.P \ fml.R \ fml.S \} \ // \text{equivalent to} \ fml3 \]

\[ fml4 = \text{slice} \ fml \ \text{including} \ \{ \ fml.E \ fml.D \ fml.F \} \]

\[ fts5 = \{ \ fml.P \ fml.W \} \ ++ \ fml.P.* \]
\[ fml5 = \text{slice} \ fml \ \text{including} \ fts5 \]
Putting all together: Example 2
From marketing, customers, product management

From existing software assets (technical variability)
From marketing, customers, product management

From existing software assets

usefulness

realizability

\[ V_1 \leftrightarrow f_1 \]
\[ V_2 \leftrightarrow f_2 \]
\[ V_3 \leftrightarrow f_3 \]

Realizability checking

1. aggregate

2. slice ("realizable part")

3. compare

4. merge diff ("unrealizable products")

\[\{V_1, V_3, V_2, VP_1\}, \{V_1, VP_1\}, \{V_3, VP_1\}, \{VP_1\}\]

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

/*
 * Metzger et al. 2007, RE'07
 * Disambiguating the ..... 
 * Figure 1, Section 3
 */

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

MacBook-Pro-de-Mathieu-2:FML-scripts macher$ java -jar -Xmx1024M ../FML-0.9.9.6.jar realizability.fml
FAMILIAR (for FeAture Model scrIpt Language for manIplementation and Automatic Reasoning) version 0.9.9.6
University of Nice Sophia Antipolis, UMR CNRS 6070, I3S Laboratory
https://nyx.unice.fr/projects/familiar/
fml> ls
(FEATURE_MODEL) gFM
(FEATURE_MODEL) fmSoftware
(FEATURE_MODEL) fmPLDiff
(FEATURE_MODEL) fmPLPrime
(FEATURE_MODEL) fmPL
(SET) xlink
fml> configs fmPLDiff
res1: (SET) {{VP1;V1};{VP1;V3};{VP1};{V3;V1;VP1;V2}}
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

Philippe Merle, software architect of FraSCAti

1 Automatic Extraction

2

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.)
#2 from product descriptions to feature models

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>Different Licences</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DrupalWiki</td>
<td>GPL2</td>
<td>PHP</td>
<td>Database</td>
<td>Community</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>

640 configurations (634 counter examples)

Exact set of configurations, each configuration corresponding to at least one product
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, \ldots, 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 directive
  – 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)
large number of software configurations for a large number of requirements
Implementation: under the hood

(a) Task model
- Task
  - Counting
  - Intrusion
- QoS
  - Quality
  - Person
- Object of interest
- Scene context
  - Scene description
  - Environment
- Precision
- Response time
  - Frame rate
- Video surveillance application

(b) Platform model
- Acquisition
- Segmentation
  - Traversal algorithm
- Kernel function
  - Grid step
  - With window
- Classification
  - Clustering
- Model
- Video surveillance platform
  - View
  - Artificial
  - Indoors
  - Camera
  - Lighting

Features:
- Mandatory feature
- Optional feature
- Alternative features (XOR)
- Or-features (OR)
- Cross-model constraint
- Internal constraint
- Specification feature imposed by the application
- Specification feature deduced from internal constraints
- Implementation feature deduced from cross-constraints (transformation)
- "Neutral" implementation feature
Putting all together: Example 5 & Demo...
Realizing a Car Crash Management System SPL

Requirements Definition Document
for a Software Product Line of Car Crash Management Systems

May 5, 2012
http://eserg0.site.uottawa.ca/cma2012

Authors:
Álfredo 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)
What’s next?

• Feature modeling with FAMILIAR
  – Applicability, Learnability, Expressiveness, Usability
  – New implementation with SCALA

• Connection of feature models to other artifacts
  – Automated product derivation
  – Verification & Certification

• Multiple Software Product Lines