Software Product Lines with Design Choices: Reasoning about Variability and Design Uncertainty

Michalis Famelis, Julia Rubin, Krzysztof Czarnecki, Rick Salay, Marsha Chechik
Software Product Line Basics

Feature model, Domain model, Feature mapping

Different feature *configurations* result in different variants

PL models a *set* of related, but different products
Washing Machine Product Line

Domain Model

Locking
- Heat
- Delay
- [heatEnabled;delayEnabled]
  / HeaterOn()
  Heat
  Heat V Delay

Unlocking
- Heat
- [not heatEnabled;not delayEnabled]
  / QuickCool()
  -Dry
  Dry

Waiting
- Heat
- Delay
- Heat V Delay

Washing
- entry/TempCheck()
  Heat

Drying
- Dry
- QuickCool()

Feature Model

Wash
- Heat
- excludes

Delay
- Heat

Dry
- Heat V Delay

Variant with Dry and Delay

Model without variability

- **Locking**
  - [delayEnabled]
  - not delayEnabled] / wash.Start();
- **Unlocking**
  - / QuickCool()
- **Waiting**
  - / wash.Start();
- **Washing**
  - / QuickCool()
- **Drying**

Configuration

- **Heat**
- **Delay**
- **Dry**
Evolving Washing Machines

**Domain Model**

- **Locking**
  - Heat
  - Delay
  - [heatEnabled; delayEnabled] / HeaterOn()
  - [not heatEnabled; not delayEnabled] / washer.Start();

- **Unlocking**
  - Heat
  - Delay
  - [not heatEnabled; not delayEnabled] / QuickCool()

- **Waiting**
  - Heat V Delay
  - [heatEnabled; delayEnabled] / HeaterOn()
  - [not heatEnabled; not delayEnabled] / heaterOff();
  - washer.Start();

- **Washing**
  - entry/TempCheck()
  - Heat

- **Drying**
  - Dry

**Decision point Mutex**
Not sure whether Heat and Delay are mutually exclusive

**Feature Model**

- **Wash**
- **Heat**
- **Delay**
- **Dry**

- Heat excludes Delay

- Heat ∨ Delay

**Decision point HasNoSpin**
Not sure whether to put a guard NoSpin on transition
Design 1: No Mutex or HasSpin

Domain Model

- **Locking**
  - Heat
  - Delay
  - [heatEnabled; delayEnabled]/ HeaterOn()
  - [not heatEnabled; not delayEnabled]/
    - wash.Start();

- **Unlocking**
  - Heat
  - Delay
  - [not heatEnabled; not delayEnabled]/
    - QuickCool()
  - Dry

- **Waiting**
  - Heat
  - Delay
  - Heat V Delay
  - [heatEnabled; delayEnabled]/
    - HeaterOn()
  - [not heatEnabled; not delayEnabled]/
    - HeaterOff();
  - Heat
  - Delay
  - Heat V Delay

- **Washing**
  - Entry/TempCheck()
  - Heat
  - Dry

- **Drying**
  - Heat
  - Delay
  - Heat V Delay

Feature Model

- Wash
- Delay
- Dry

Heat and Delay not mutually exclusive

No guard on the transition
Design 2: Mutex and HasSpin

Domain Model

- **Locking**
  - Heat
  - [heatEnabled; delayEnabled]
    - / HeaterOn()
    - Heat
    - Heat V Delay
  - Delay
    - Heat V Delay

- **Unlocking**
  - Dry
    - / QuickCool()
  - Dry
    - / QuickCool()

- **Waiting**
  - Heat
    - / HeaterOn()
    - Heat
    - Heat V Delay
  - Delay
    - Heat V Delay
  - Heat V Delay
  - Heat V Delay
  - / HeaterOff();
  - / QuickCool();

- **Washing**
  - entry/TempCheck()
  - Heat
  - [NoSpin]
  - NoSpin guard on the transition

Feature Model

- **Heat**
- **Delay**
- **Dry**

Heat excludes Delay
Heat ∨ Delay
Heat ∨ Delay
Heat ∨ Delay
Heat ∨ Delay
Heat

Dry
 Dry

NoSpin guard on the transition
A Design Space of SPLs

Which one should we choose?

To explore this space, we need its properties

\[
\{\text{Mutex, HasNoSpin}\} \\
\{\neg\text{Mutex, HasNoSpin}\} \\
\{\text{Mutex, \neg HasNoSpin}\} \\
\{\neg\text{Mutex, \neg HasNoSpin}\}
\]
Motivation

How to model this design space?

What are relevant properties for exploring it? How to check them?

What to do when properties are violated?
Design choices can affect the Feature model, Domain model, Feature mapping

They define a design space of product lines

Given a space of product lines, which one should be selected (and why)?

Feature combinations produce possible products

Decision combinations produce possible PLs
Two Different Kinds of Choices

<table>
<thead>
<tr>
<th>Reason</th>
<th>Variability</th>
<th>Design Uncertainty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Market demands</td>
<td>Market demands for product variants</td>
<td>Incomplete information, design alternatives, stakeholder conflicts, etc.</td>
</tr>
<tr>
<td>Reason</td>
<td>Features</td>
<td>Decisions</td>
</tr>
<tr>
<td>Granularity</td>
<td>Features</td>
<td>Decisions</td>
</tr>
<tr>
<td>Expression</td>
<td>Product line (PL) models</td>
<td>Partial models</td>
</tr>
<tr>
<td>Semantics</td>
<td>Set of artifacts produced by combinations of features</td>
<td>Set of artifacts produced by combinations of decisions</td>
</tr>
<tr>
<td>Horizon</td>
<td>Long term</td>
<td>Short term</td>
</tr>
</tbody>
</table>
Modelling SPLDCs

Potential candidates: 

**Clafer**: textual structural modelling language with native variability abstractions  
**PEnP**: product line implementation in Jetbrains MPS  
**MU-MMINT**: implementation of partial modelling in Eclipse  

Main challenges:  
Technological integration of tools  
Identification of usable syntax for intuitively expressing the two concerns
Software Product Lines with Design Choices

Motivation

How to model this design space?

What are relevant properties for exploring it? How to check them?

What to do when properties are violated?
A Design Space of Product Lines

Goal: Use properties to explore the design space of PL’s

Decision combinations produce possible PLs
e.g., {¬Mutex, HasSpin}

Feature combinations produce possible products
e.g., {¬Delay, Heat, Dry}
Constraining the Design Space using Properties

For a product-level property P, we define four SPLDC-level properties using the modalities:

- Use **All** for critical properties and **Some** for desirable properties
- Use **Necessary** when you are sure it is needed and **Possible** when unsure but don’t want to exclude the possibility

<table>
<thead>
<tr>
<th></th>
<th>Necessary for the product line</th>
<th>Possible for the product line</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>All products have P</strong></td>
<td>All products in All product lines</td>
<td>All products in Some product line</td>
</tr>
<tr>
<td><strong>Some products have P</strong></td>
<td>Some product in All product lines</td>
<td>Some product in Some product line</td>
</tr>
</tbody>
</table>
Necessary-Some (NS)

All product lines

Some product
Possible-Some (PS)

Some product line

Some product...

...
Possible-All (PA)

Some product line

All products
Necessary-All (NA)

All product lines

All product

...
Necessary-All (NA)

Example product-level property: State **Unlocking** must always be reached
- a washing machine that violates this is unacceptable

Example Analysis
Check: does the SPLDC-level NA property hold?
Uncertainty in a Washing Machine Product Line

Domain Model

Decision point **Mutex**
Not sure whether Heat and Delay are mutually exclusive.

[heatEnabled;delayEnabled] / HeaterOn()

Heat

Delay

Heat V Delay

Waiting

NO!

If $\text{HasSpin} = \text{true}$ and $\text{Dry} = \text{true}$ then the guard may prevent state Unlocking from being reached.

Feature Model

Heat

Delay

Dry

Wash

Delay

Dry

Decision point **HasNoSpin**
Not sure whether to put a guard NoSpin on transition.
SPLDC-level Properties

SPLDC property modalities apply to any kind of product-level properties
   Behavioural (e.g., temporal logic)
   Structural (e.g., well-formedness constraints)
   Other (e.g., non-functional properties)

The main challenge is in implementing the lifted analysis
   such that it generates separate feedback for variability and design uncertainty
Generating Feedback for NA Property

(Necessary-All)

Counter-example product line

Counter-example product
Software Product Lines with Design Choices

Motivation

How to model this design space?

What are relevant properties for exploring it? How to check them?

What to do when properties are violated?
Responses to Property Violation

Necessary-All?  ✗
Response 1: Relax Constraint

Necessary-All? \(\times\)
Possible-All? \(\checkmark\)
Response 2: Reduce Uncertainty

Necessary-All?  ❌

Reduce Uncertainty

e.g., decide $HasSpin = false$
Response: Reduce Uncertainty

Necessary-All? ☑
Response: Reduce Variability

Necessary-All? ✗

Reduce Variability

e.g., remove feature *Dry*
Response 3: Reduce Variability

Necessary-All?  ✔️
Responding to SPLDC Property Violations

Guidance for leveraging generated feedback:
Should the problem be addressed at the SPLDC level?
If so, which strategy is most appropriate?
If not, can we recommend repairs?

Sometimes we can address problems at the SPLDC level by **addition**
E.g., adding previously forbidden configuration options
Can we recommend such additions?

Necessary-Some (NS) violated:
Software Product Lines with Design Choices

Motivation

How to model this design space?

What are relevant properties for exploring it?
How to check them?

What to do when properties are violated?
Current Work

Focus on structural SPLDC properties
SPLDCs as first-order theories
Preliminary specification using Alloy
Investigating more sophisticated reasoning engines (e.g. QBF, Second Order Logic)

Next Steps

Evaluating scalability of the tools and effectiveness of the methodology
Better automation of response strategies
Case studies
  * Power Window, “Real” Washing Machine, GitHub clones of configurable projects
Design Uncertainty and Variability

Similar but different concerns
Can be used together to model and explore a space of SPL designs

We defined:
Four natural classes of SPLDC-level properties that can be checked
Strategies for responding to property violations

Decision combinations produce possible PLs
Feature combinations produce possible products

Prod_1
Prod_2
Prod_n