ABS support for SPLs and Multi SPLs

Ferruccio Damiani[1], Reiner Hähnle[2], Eduard Kamburjan[2], Michael Lienhardt[1]

[1] Università di Torino
[2] TU Darmstadt

Outline

1. Background, motivation and challenge
2. Proposal overview
3. A step further

Outline

Multi SPL (MPL)

MPL

- a set of SPLs
  - self-contained
  - interdependent
- managed and developed in a decentralized fashion
- by multiple teams and stakeholders
- represents a large-scale or ultra-large-scale system
- ...

Multi SPL (MPL)

MPL

- a set of SPLs
  - self-contained
  - interdependent
- managed and developed in a decentralized fashion
- by multiple teams and stakeholders
- represents a large-scale or ultra-large-scale system
- ...

A survey

- G. Holl, P. Grünbacher, R. Rabiser, A systematic review and an expert survey on capabilities supporting multi product lines, Information & Software Technology 54 (8) (2012) 828-852. doi:10.1016/j.infsof.2012.02.002.

The FormbaR use case

Modeling railway operations (in ABS)

- many variants of signals or switches
- all used on the same track
- need to interoperate

The FormbaR use case

Modeling railway operations (in ABS)

- many variants of signals or switches
- all used on the same track
- need to interoperate

General requirement for language support:

- use multiple variants (possibly from the same MPL) in one single application

SPLs in ABS

Delta-Oriented Programming (DOP)

- Feature model
  - a set of features (F)
  - a formula over the set of features (describes the set of products, i.e., an element of 2^F)
- Artifact base
  - a base program (a Core ABS program)
  - a set of deltas (describe changes to a Core ABS program)
- Configuration knowledge
  - Activation mapping (associates each delta to an activation condition)
  - Application order (a partial order between deltas)

SPLs in ABS

Delta-Oriented Programming (DOP)

- Feature model
  - a set of features (F)
  - a formula over the set of features (describes the set of products, i.e., an element of 2^F)
- Artifact base
  - a base program (a Core ABS program)
  - a set of deltas (describe changes to a Core ABS program)
- Configuration knowledge
  - Activation mapping (associates each delta to an activation condition)
  - Application order (a partial order between deltas)

Variant generation: given a product, a variant is generated by

- applying the activated deltas
  - to the base program
  - according to the application order

Analysis of ABS SPLs

Several analyses

- On the feature model
  - Empty feature model, dead features, false-optional features,...
- On the artifact base
  - Type uniformity,...
- On the configuration knowledge
  - Dead deltas,...
- On the whole SPL
  - Type safety, useless code,...

Analysis of ABS SPLs

Several analyses

- On the feature model
  - Empty feature model, dead features, false-optional features,...
- On the artifact base
  - Type uniformity,...
- On the configuration knowledge
  - Dead deltas,...
- On the whole SPL
  - Type safety, useless code,...

An ABS SPL is type safe iff for each product the corresponding variant

- can be generated
- is a well-typed Core ABS program

The challenge

Design language constructs to support MPLs—requirements:

1. expressiveness: use multiple variants (possibly from the same MPL) in one single application
   - E.g.: capture the `FormaR` use case
2. usability: analyses are feasible
3. ...

The challenge

Design language constructs to support MPLs—requirements:

1. expressiveness: use multiple variants (possibly from the same MPL) in one single application
   - E.g.: capture the `FormaR` use case
2. usability: analyses are feasible
3. ...

The challenge:

- Ferruccio Damiani, Reiner Hähnle, Eduard Kamburjan, Michael Lienhardt. Interoperability of Software Product Line Variants. SPLC 2018 Challenge Track. `http://splc2018.net/call-for-papers/call-for-challenge-solutions/`

The challenge

Design language constructs to support MPLs—requirements:

1. expressiveness: use multiple variants (possibly from the same MPL) in one single application
   - E.g.: capture the `FormaR` use case
2. usability: analyses are feasible
3. ...

The challenge:

- Ferruccio Damiani, Reiner Hähnle, Eduard Kamburjan, Michael Lienhardt. Interoperability of Software Product Line Variants. SPLC 2018 Challenge Track. `http://splc2018.net/call-for-papers/call-for-challenge-solutions/`

This talk:

- addressing the challenge by extending ABS to support MPLs

Outline

1. Background, motivation and challenge
2. Proposal overview
3. A step further

Methodology

1. Design foundational calculi (to capture/model existing relevant notions)
   - Featherweight Core ABS with Modules (FAM)
   - Featherweight Delta ABS with Modules (FDABS)

2. Design minimal extensions of the calculi (to capture/model new relevant notions)
   - Develop examples
   - Identify and prove properties

3. Asses by
   - Implementation
   - Case studies

FAM syntax

$\text{Prgm} ::= \overline{\text{Mod}}$	Program
$\text{Mod} ::= \textbf{module}\ \text{M};\ \textbf{import}\ \text{SC}\ \textbf{from}\ \text{M};\ \textbf{export}\ \text{SC};\ \overline{\text{CD}}\ \overline{\text{ID}}$	Module
$\text{SC} ::= \overline{\text{C}}, \overline{\text{I}} \mid *$ $\text{CD} ::= \textbf{class}\ \text{C}\ [\textbf{implements}\ \text{IR}]\ \{\overline{\text{MD}}\ \overline{\text{FD}}\}$	Selection, Class
$\text{CR} ::= \text{M.C} \mid \text{C}$ $\text{IR} ::= \text{M.I} \mid \text{I}$	Class/Interface Reference
$\text{FD} ::= \text{T f=e}$ $\text{MD} ::= \text{MSD}\{s\}$	Field, Method
$\text{MSD} ::= \text{T m}(\overline{\text{T v}})$ $\text{ID} ::= \textbf{interface}\ \text{I}\ [\textbf{extends}\ \text{IR}\ \overline{\text{IR}}]\ \{\overline{\text{MSD}}\}$	Signature, Interface
$\text{e} ::= \textbf{new}\ \text{CR}(\overline{\text{e}}) \mid \ldots$ $\text{T} ::= \text{IR} \mid \texttt{Unit} \mid \texttt{Int} \mid \cdots$	Expression, Type

Variant Interoperable SPLs (VPLs)

An MPL is

- A set of Variant Interoperable SPLs (VPLs)

  (i.e., FDAM SPLs where the base program contains a unique part)
- A glue program

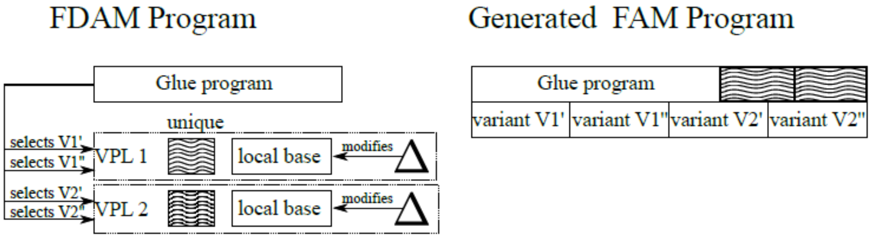
  (i.e., a FAM program containing variant references)

Variant Interoperable SPLs (VPLs)

An MPL is

- A set of Variant Interoperable SPLs (VPLs)
  (i.e., FDAM SPLs where the base program contains a unique part)
- A glue program
  (i.e., a FAM program containing variant references)

Variant generation illustration:



FDAM syntax and extended references syntax

Vpl ::=**productline** V; **features** $\overline{\mathsf{F}}$ **with** φ; Prgm **unique** $\overline{\mathsf{Mod}}$ $\overline{\Delta}$ DConfig				VPL
Δ ::=**delta** D;$\overline{\mathsf{CO}}$ $\overline{\mathsf{IO}}$	DConfig ::= $\overline{\mathsf{SCO}}$	SCO ::= **delta** D **when** φ;		Delta
CO ::=AO | MO | RO | **uses** M	IO ::= IAO | IMO | IRO | **uses** M			Configuration
AO ::=**adds** CD	RO ::= **removes** CR;	MO ::= **modifies** CR{CIO}		Class Operation
CIO ::=**adds** CIC | **removes** CIC	CIC ::= FD | MD			Inner Operation
IAO ::=**adds** ID	IRO ::= **removes** IR;	IMO ::= **modifies** IR{IIO}		Interface Operation
IIO ::=**adds** MSD | **removes** MSD				Inner Operation

CR ::=VR.M.C | M.C | C	IR ::= VR.M.I | M.I | I	Extended Class/Interface Reference
VR ::=V | V[$\overline{\mathsf{F}}$]		Variant Reference

Example: the VPL SLine (configuration knowledge is omitted) and a glue program

```
productline SLine;
features Main, Pre, Light, Form with Main↔ ¬Pre ∧ Light↔ ¬Form;
module SMd;
class Signal implements SLine.SMd.Sig {}
unique{
  module SMd;
  interface Sig { ... }
}
delta SigForm;  modifies class SMd.Signal { ... } ...
delta SigPre;   modifies class SMd.Signal { ... } ...
delta SigMain;  modifies class SMd.Signal { ... } ...
delta SigLight; modifies class SMd.Signal { ... } ...

module main;
class Main{
  Unit main() {
    SLine.SMd.Sig s1 = new SLine[Pre,Form].SMd.Signal();
    SLine.SMd.Sig s2 = new SLine[Main,Form].SMd.Signal();
    s1.connect(s2);
    SLine.SMd.Sig s3 = new SLine[Pre,Form].SMd.Signal();
    SLine.SMd.Sig s4 = new SLine[Main,Form].SMd.Signal();
    s3.connect(s4);
  }
}
```

Dependend VPLs (DVPLs)

$$\begin{array}{ll} \mathsf{Dvpl} ::= \textbf{productline}\ \mathsf{V}(\overline{\mathsf{V}\ \mathsf{P}});[\textbf{uses}\ \overline{\mathsf{V}};]\textbf{features}\ \overline{\mathsf{F}}\ \textbf{with}\ \psi; & \text{DVPL} \\ \quad \mathsf{Prgm}\ \textbf{unique}\{\overline{\mathsf{Mod}}\}\ \overline{\Delta}\ \mathsf{DConfig} & \end{array}$$

$$\mathsf{VR} ::= \mathsf{V} \mid \mathsf{V}[\overline{\mathsf{F}}](\mathsf{VR}) \mid \mathsf{P} \qquad \text{Variant References}$$

Example: the DVPL BlocklineLine (version 1)

```
productline BlockLine;
uses SignalLine;
unique{
module BMd;
interface Block{
    addSignal(SignalLine.SMd.Sig sig);
}
...
```

Example: the DVPL BlocklineLine (version 2)

```
productline BlockLine(SLine sl1, SLine sl2);
features Light, Form with sl1.Form↔sl2.Form ∧ sl1.Pre
           ∧ sl2.Main ∧ Light ↔ sl1.Light ∧ Form ↔ sl1.Form;
delta AlwaysDelta;
adds class Block{
    SLine.SMd.Sig s1 = new sl1.SMd.Signal();
    SLine.SMd.Sig s2 = new sl2.SMd.Signal();
    SLine.SMd.Sig s3 = new sl1.SMd.Signal();
    SLine.SMd.Sig s4 = new sl2.SMd.Signal();
    Unit Block(){
        s1.connect(s2);
        s3.connect(s4);
    }
}
delta AlwaysDelta when True;
```

```
BlockLine[Light](SLine[Light, Pre](), SLine[Light, Main]())
```

Example: the DVPL LineLine

```
productline LineLine(BlockLine bl1, BlockLine bl2);
delta AlwaysDelta;
adds class LMd.Line {
    bl1.BMd.Block b1 = new bl1.BMd.BlockStelle();
    bl2.BMd.Block b2 = new bl2.BMd.BlockStelle();
    SigLine.SMd.Sig s1 = b1.BMd.getRightSignal();
    SigLine.SMd.Sig s2 = b2.BMd.getLeftSignal();
    Unit Line(){
        b1.connect(s2);
        b2.connect(s1);
    }
}
delta AlwaysDelta when True;
```

Outline

Decoupling DVPLs

Consider

- Feature Model interface relation: $\mathcal{M}'$ is an interface of $\mathcal{M}$ iff

  $\mathcal{M}'$ is obtained from $\mathcal{M}$ by dropping some feature

Decoupling DVPLs

Consider

- Feature Model interface relation: $\mathcal{M}'$ is an interface of $\mathcal{M}$ iff

  $\mathcal{M}'$ is obtained from $\mathcal{M}$ by dropping some feature

from

- M. Acher, P. Collet, P. Lahire, and R. B. France. Slicing feature models. In 26th IEEE/ACM International Conference on Automated Software Engineering, (ASE), 2011, pages 424-427, 2011.
- R. Schröter, S. Krieter, T. Thüm, F. Benduhn, G. Saake. Feature model interfaces: The highway to compositional analyses of highly congurable systems, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, 2016, pp. 667-678. doi:10.1145/2884781.2884823.

Decoupling DVPLs

Consider

- Feature Model interface relation: $\mathcal{M}'$ is an interface of $\mathcal{M}$ iff

  $\mathcal{M}'$ is obtained from $\mathcal{M}$ by dropping some feature

from

- M. Acher, P. Collet, P. Lahire, and R. B. France. Slicing feature models. In 26th IEEE/ACM International Conference on Automated Software Engineering, (ASE), 2011, pages 424-427, 2011.
- R. Schröter, S. Krieter, T. Thüm, F. Benduhn, G. Saake. Feature model interfaces: The highway to compositional analyses of highly congurable systems, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, 2016, pp. 667-678. doi:10.1145/2884781.2884823.

and lift it to (D)VPLs

Program signatures and Program interface relation

Define

- Program signature: a program deprived of methods' bodies
- Program interface relation: a program signature PS is a interface of a program P iff

  P provides all the classes, interfaces, fields, methods and subtyping relations declared in PS

VPL signatures and VPL interface relation

Define

- VPL signature (VPLS): a VPL deprived of methods' bodies
- VPL interface relation: a VPLS K is a interface of a VPL V iff
  1. the feature model of K is an interface of the feature model of V
  2. for each product q of K and all its completions p in V

     the variant PS for q in K is an interface of the variant P for p in V

VPL signatures and VPL interface relation

Define

- VPL signature (VPLS): a VPL deprived of methods' bodies
- VPL interface relation: a VPLS K is a interface of a VPL V iff
  1. the feature model of K is an interface of the feature model of V
  2. for each product q of K and all its completions p in V

     the variant PS for q in K is an interface of the variant P for p in V

building on

- Ferruccio Damiani, Michael Lienhardt, Luca Paolini. A Formal Model for Multi SPLs. 7th International Conference on Fundamentals of Software Engineering (FSEN), Vol. 10522 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2017, pp. 67-83. doi: 10.1007/978-3-319-68972-2_5