ABS support for SPLs and Multi SPLs

Ferruccio Damiani\textsuperscript{1}, Reiner Hähnle\textsuperscript{2}, Eduard Kamburjan\textsuperscript{2}, Michael Lienhardt\textsuperscript{1}

\textsuperscript{1} Università di Torino
\textsuperscript{2} TU Darmstadt
Outline

1. Background, motivation and challenge

2. Proposal overview

3. A step further
Outline

1. Background, motivation and challenge

2. Proposal overview

3. A step further
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
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**
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:

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:


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. Assess by
   - Implementation
   - Case studies
FAM syntax

\[ \text{Prgm ::= Mod} \]
\[ \text{Mod ::= module M; import SC from M; export SC; CD ID} \]
\[ \text{SC ::= } \overline{C}, \overline{I} | * \]
\[ \text{CD ::= class C [implements IR] \{MD FD\}} \]
\[ \text{CR ::= M.C | C} \]
\[ \text{IR ::= M.I | I} \]
\[ \text{FD ::= T f=e} \]
\[ \text{MD ::= MSD\{s\}} \]
\[ \text{MSD ::= T m(T,v)} \]
\[ \text{ID ::= interface I [extends IR IR] \{MSD\}} \]
\[ \text{e ::= new CR(e) | ...} \]
\[ \text{T ::= IR | Unit | Int | ...} \]
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

\[ Vpl ::= \text{productline } V; \text{ features } \overline{F} \text{ with } \varphi; \]

\[ \text{Prgm unique } \overline{Mod} \overline{\Delta} \text{ DConfig} \]

\[ \Delta ::= \text{delta } D; \overline{CO} \overline{IO} \quad \text{DConfig} ::= \overline{SCO} \quad \text{SCO} ::= \text{delta } D \text{ when } \varphi; \]

\[ \text{CO} ::= \text{AO } | \text{ MO } | \text{ RO } | \text{ uses M } \quad \text{IO} ::= \text{ IAO } | \text{ IMO } | \text{ IRO } | \text{ uses M } \]

\[ \text{AO} ::= \text{adds } CD \quad \text{RO} ::= \text{removes } CR; \quad \text{MO} ::= \text{modifies } CR\{\text{CIO}\} \]

\[ \text{CIO} ::= \text{adds } CIC | \text{removes } CIC \quad \text{CIC} ::= \text{FD } | \text{ MD} \]

\[ \text{IAO} ::= \text{adds } ID \quad \text{IRO} ::= \text{removes } IR; \quad \text{IMO} ::= \text{modifies } IR\{\text{IIO}\} \]

\[ \text{IIO} ::= \text{adds } MSD | \text{removes } MSD \]

\[ \text{CR} ::= \text{VR.M.C } | \text{ M.C } | \text{ C } \quad \text{IR} ::= \text{VR.M.I } | \text{ M.I } | \text{ I } \]

\[ \text{VR} ::= V | V[\overline{F}] \]

Extended Class/Interface Reference

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

```java
1 productline SLine;
2 features Main, Pre, Light, Form with Main ↔ ¬Pre ∧ Light ↔ ¬Form;
3 module SMd;
4 class Signal implements SLine.SMd.Sig {}
5 unique{
6   module SMd;
7     interface Sig { ... }
8 }
9 delta SigForm; modifies class SMd.Signal { ... } ...
10 delta SigPre; modifies class SMd.Signal { ... } ...
11 delta SigMain; modifies class SMd.Signal { ... } ...
12 delta SigLight; modifies class SMd.Signal { ... } ...
13
14 module main;
15 class Main{
16   Unit main() {
17     SLine.SMd.Sig s1 = new SLine[Pre,Form].SMd.Signal();
18     SLine.SMd.Sig s2 = new SLine[Main,Form].SMd.Signal();
19     s1.connect(s2);
20     SLine.SMd.Sig s3 = new SLine[Pre,Form].SMd.Signal();
21     SLine.SMd.Sig s4 = new SLine[Main,Form].SMd.Signal();
22     s3.connect(s4);
23   }
24 }
```
Dependend VPLs (DVPLs)

\[
\text{Dvpl} ::= \text{productline } V(\overline{V P}); \text{[uses } \overline{V};\text{]} \text{features } \overline{F} \text{ with } \varphi; \quad \text{DVPL}
\]

\[
\text{Prgm unique}\{\text{Mod}\} \Delta \text{DConfig}
\]

\[
\text{VR} ::= V \mid V[\overline{F}](\text{VR}) \mid P \quad \text{Variant References}
\]
Example: the DVPL BlockLineLine (version 1)

```plaintext
1 productline BlockLine;
2 uses SignalLine;
3 unique{
4 module BMd;
5 interface Block{
6   addSignal(SignalLine.SMd.Sig sig);
7 }
8 ...}
```
Example: the DVPL BlocklineLine (version 2)

```java
1. productline BlockLine(SLine sl1, SLine sl2);
2. features Light, Form with sl1.Form ↔ sl2.Form ∧ sl1.Pre
     ∧ sl2.Main ∧ Light ↔ sl1.Light ∧ Form ↔ sl1.Form;
3. delta AlwaysDelta;
4. adds class Block{
5.     SLine.SMd.Sig s1 = new sl1.SMd.Signal();
6.     SLine.SMd.Sig s2 = new sl2.SMd.Signal();
7.     SLine.SMd.Sig s3 = new sl1.SMd.Signal();
8.     SLine.SMd.Sig s4 = new sl2.SMd.Signal();
9.     Unit Block(){
10.         s1.connect(s2);
11.         s3.connect(s4);
12.     }
13. }
14. }
15. delta AlwaysDelta when True;

BlockLine[Light](SLine[Light, Pre](), SLine[Light, Main]())
```
```java
26 productline LineLine(BlockLine bl1, BlockLine bl2);
27 delta AlwaysDelta;
28 adds class LMd.Line {
29 b1.BMd.Block b1 = new b1.BMd.BlockStelle();
30 b2.BMd.Block b2 = new b2.BMd.BlockStelle();
31 SigLine.SMd.Sig s1 = b1.BMd.getRightSignal();
32 SigLine.SMd.Sig s2 = b2.BMd.getLeftSignal();
33 Unit Line(){
34 b1.connect(s2);
35 b2.connect(s1);
36 }
37 }
38 delta AlwaysDelta when True;
```
Outline

1. Background, motivation and challenge

2. Proposal overview

3. A step further
Decoupling DVPLs

Consider

- **Feature Model interface relation:** $M'$ is an interface of $M$ iff $M'$ is obtained from $M$ 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


Decoupling DVPLs

Consider

- **Feature Model interface relation:** $M'$ is an interface of $M$ iff $M'$ is obtained from $M$ by dropping some feature

and lift it to (D)VPLs

---


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 $PS$
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