Modellizzazione Orientata alle Differenze: un approccio per descrivere la variabilità nel software e nei sistemi

Ferruccio Damiani*
University of Turin - Department of Computer Science
http://www.unito.it/~damiani http://di.unito.it/movere

5 febbraio 2019, Torino

*: Laurea: Laboratorio di Programmazione II
Laurea Magistrale: Programmazione per Dispositivi Mobili
Software Product Lines (SPLs)

Highly-configurable software systems can be described as SPLs.

An SPL is a family of similar programs, called variants, that
  • have a well-documented variability and
  • are generated from a common artifact base.

EXAMPLES: see the Product Line Hall of Fame --- http://splc.net/hall-of-fame/
An SPL consists of:

1. a **feature model**: specifies the set of variants in terms of **features** (each variant is identified by a set of features, called a **product**)
2. an **artifact base**: reusable code artifacts used to build the variants
3. **configuration knowledge**: connects feature model and artifact base
Feature Model


EXAMPLE.

Feature Diagram:

Logical representation: \( v \) and \( b \) and (\( s \) or \( t \))

Extensional representation:

\[
F = \{v, b, s, t, f, c\}
\]

\[
\Lambda = \{\{v, b, t\}, \{v, b, t, f\}, \{v, b, t, c\}, \{v, b, t, f, c\}, \{v, b, s\}, \{v, b, s, f\}, \{v, b, s, c\}, \{v, b, s, f, c\}, \{v, b, s, t\}, \{v, b, s, t, f\}, \\
\{v, b, s, t, c\}, \{v, b, s, t, f, c\}\}
\]

• 6 features (2 abstract, 4 concrete)
• 12 products

In general: given \( n \) features, up to \( 2^n \) products!
Orders of Magnitude

33 Features
one configuration for every single person on the earth!

320 Features
more configurations than atoms existing in the universe!
Artifact base and configuration knowledge: annotative approach

The artifact base is an annotated program.

Example (older than the notion of SPL): C preprocessor annotations.

```
#define FEATURE
#if defined FEATURE
 /* code */
#endif
#endif
```

- Configuration knowledge and artifact base are deeply entangled.
- Difficult to maintain.
The SPL analysis issue


Three approaches to SPL analysis:

- **Product based**: generate and analyze each variant in isolation.
- **Feature based**: the artifact base is analyzed in isolation without considering feature model and configuration knowledge.
- **Family based**: analyze the artifact base, without generating any variant, by exploiting feature model and configuration knowledge to derive results about all variants.

Do you see the pros and cons?
Artifact base and configuration knowledge: compositional approach

Example: **Feature Oriented Programming (FOP)**.

- **Configuration knowledge**: a total order on the features, a feature module for each feature

- **Artifact base**: feature modules can (for Java)
  - Add classes/interfaces
  - Modify classes, i.e.:
    - add fields/methods
    - modify extends and implements clauses
    - modify methods (replace bodies and possibly call the original version)
  - Modify interfaces...

FOP is well suited for feature based and family based SPL analyses.
The feature interaction problem

A feature interaction (FI) occurs when the integration of two features modifies the behavior of one or both features.

A classic example:
- the two features
  - Fire control
  - Flood control

To manage FI in FOP, you have to introduce auxiliary features.
The adoption barrier issue


Three approaches to SPL Engineering (SPLE):

• **Proactive:** the set of products to be developed is analyzed beforehand and all reusable artifacts are planned and developed in advance.

• **Reactive:** start with an initial SPL that is evolved in order to deal with changing customer requirements.

• **Extractive:** start with a set of existing legacy applications from which the other products of the SPL are generated.

FOP is well suited for proactive SPLE and, to some extent, can support reactive SPLE.
Artifact base and configuration knowledge: transformational approach

Example: *Delta Oriented Programming (DOP).*

- **Configuration knowledge:** a partial order on the features, *delta modules (deltas)* each one activated by a set of products.

- **Artifact base: deltas** can (for Java):
  - Add/remove classes/interfaces
  - Modify classes, i.e.:
    - Add/remove fields/methods
    - modify extends and implements clauses
  - Modify ...
  - Modify interfaces...

DOP allows to address FI without introducing auxiliary and supports proactive/reactive/extractive SPL.

Ina Schaefer,
TU Kaiserlautern,
Germany.
Research on DOP started in spring 2009.

The first three papers on DOP:
- Ina Schaefer, Alexander Worret, Arnd Poetzsch-Heffter: A Model-Based Framework for Automated Product Derivation. MAPLE@SPLC 2009

• First discussions (probably) around 2007.
• An European Project:
  **Highly Adaptable and Trustworthy Software using Formal Models (HATS) ---** https://www.hats-project.eu
  March 2009 - February 2013

**ABS has been developed within the HATS project.**
What is ABS about?

Consequences of design time decisions often realized only at runtime

- Modern software development often model-/feature-driven
- Most modeling languages do not address behavior rigorously
- Mismatch among artefacts from analysis and coding phases
- Complicating factors: variability, concurrency
ABS is and Executable Modeling Language, designed with analysis/code generation tools in mind

- Expressivity carefully traded off with analyzability
  - enables incremental/compositional static and dynamic analyses

- State-of-art programming language concepts
  - ADTs + functions + objects
  - type-safety by design
  - modules, components

- Layered concurrency model
  - Upper tier: asynchronous, no shared state, actor-based
  - Lower tier: synchronous, shared state, cooperative multitasking

- Modeling of variability with first-class language support
  - DOP
ABS for modeling the cloud

Reiner Hähnle, Chalmers University of Technology, Sweden.
Einar Broch Johnsen, University of Oslo, Norway.

and others...

• Another European Project:
  **Engineering Virtualized Services (Envisage)** --- [http://www.envisage-project.eu](http://www.envisage-project.eu)
  October 2013 - September 2016
Multi SPLs (MPLs)


Modern software systems often out-grow the scale of SPLs

An MPL is a set of interdependent SPLs that need to be managed in a decentralized fashion by multiple teams and stakeholders.
Extending ABS to support MPLs

• Another European Project: Scalable Hybrid Variability for Distributed Evolving Software Systems (HyVar) --- http://www.hyvar-project.eu

February 2015 - January 2018

Some HyVar papers:
Ongoing Work: Extending ABS

1. To support MPLs
   • case studies from the FormbaR project --- https://formbar.raillab.de/en/ that aims to provide a uniform model of operational and technical rulebooks for railroad operations.

2. To model Cloud/Fog/Edge/IoT (cf. the talk by Giorgio Audrito)

In April 2019 Violet Ka I Pun ( ), associate professor at:
• Western Norway University of Applied Sciences, and
• University of Oslo
will visit di.unito.it and give a talk on ABS.
Thanks!

People currently working on this topic at di.unito.it:
- Ferruccio Damiani (contact) --- http://www.di.unito.it/~damiani
- Simone Donetti
- Luca Paolini --- http://www.di.unito.it/~paolini
- Gianluca Torta --- http://www.di.unito.it/~torta

Research group:
- System Modelling, Verification and Reuse (MoVeRe) --- http://di.unito.it/movere