Tailoring Dynamic Software Product Lines

Marko Rosenmüller, Norbert Siegmund, Mario Pukall
University of Magdeburg, Germany

Sven Apel
University of Passau, Germany

GPCE'11
October 22, 2011
Software Product Lines (SPLs)

Feature-oriented Software Development (FOSD)

- Idea: generate tailor-made programs from a common code base
- Describe variability of an SPL in a feature model

![Feature Model Diagram]

Feature-oriented Programming (FOP)

- Implement each feature as a separate feature module
- Derive a program by composing feature modules according to a user-defined feature selection
Motivation

Goal: Generate DSPL

Part 1

Feature Binding

SPL

Examples: FOP, C Preprocessor
Bind features statically

Program
Compile and run program

Compile-time

Goal: Generate DSPL

DSPL

Example: components
Bind features dynamically

Running Program
Adapt at runtime

Load-time / Runtime

Part 2
Part 1: Generating Tailor-made DSPLs
Generating Tailor-made DSPLs

Feature Binding

SPL

Binding at compile-time

Binding at load-time

Binding at runtime

Generating tailor-made DSPLs

Running Program

Compile-time
Load-time
Runtime
**Dynamic Binding Units**

1. Remove unneeded features
2. Choose binding time per feature
3. Statically generate *dynamic binding units*

**SPL**

**DSPL**

**Program**

**Features**

- **A**
- **B**
- **C**
- **D**

**Binding Units**

**Compose binding units**

**Adapt at runtime**

**Compile-time** | **Load-time** | **Runtime**

Marko Rosenmüller, Norbert Siegmund, Mario Pukall, Sven Apel
Generating a DSPL with FOP

For each compound feature:
- Merge Compound Feature
- Remove dead features
- Transform model constraints

For each binding unit:
- Compose Binding Unit
- Generate interface and binding code
- Compile binding unit (e.g., into DLL)

Dynamic binding and runtime adaptation using the DSPL’s feature model
Tailoring Dynamic Software Product Lines

Feature Model Transformation

Model transformation

Compound Features
- **CORE:** DB’ ↔ Index ↔ BTree
- **QE:** QUERYENGINE
- **TXN:** TRANSACTION ↔ LOGGING

Refactoring

Simplification
Part 2: Runtime Adaptation
Runtime Adaptation with FeatureAce

FeatureAce
(uses a SAT solver)

Feature Model

Adaptation Rules

Meta-program (feature-based composition)

Meta-level

Base-level

Base-program (binding units)
Feature-based Adaptation

Adaptation Rules (defined w.r.t. SPL features)

- Declarative description of actions that add and remove feature model constraints (i.e., configuration constraints)
  - Arbitrary propositional formulas over SPL features

```
OnFrequentRead
=> addConstraint(Btree v Hash)
```
Feature Model Transformation

Model transformation

addConstraint (Btree v Hash)

Compound Features
- **CORE**: DB' ↔ INDEX ↔ BTREE
- **QE**: QUERYENGINE
- **TXN**: TRANSACTION ↔ LOGGING

addConstraint (Btree v false)

Refactoring

addConstraint (Btree)

Simplification

addConstraint (DB')
Adaptation Safety

**DSPL Feature Model**
- Corresponds to actual dynamic variability
- Is a specialization of the SPL’s feature model
- SPL type system: guarantees type safety of all variants

**Correctness of Adaptation Rules**
- Detect rules that are invalid w.r.t. the DSPL’s feature model using a SAT solver:
  For every rule $R$ of a DSPL: $\text{SAT}(\text{constraints}(R) \land \text{FM}_{\text{DSPL}})$
Evaluation

- **Implemented with FeatureC++**  ([http://fosd.de/fcc](http://fosd.de/fcc))
- **Case Study**: an SPL for data management in sensor networks
  - Adapt query processing at runtime
Conclusion

**Generate Tailor-made DSPLs**

- Include only required features
- User-defined binding units
- Customizable adaptation infrastructure
- Fine-grained static customization
- Reduced resource consumption and overhead for dynamic binding
- Minimized complexity of adaptation

**Feature-based Runtime Adaptation**

- Describe adaptations w.r.t. SPL features
  - Independent of application scenario
  - Customize adaptation rules (see paper)
- Safe adaptation using DSPL features and SAT solver