Generative Software Product Line Development using Variability-Aware Design Patterns

Christoph Seidl, Sven Schuster, Ina Schaefer
May 16, 2015

Sven Schuster's Presentation

Software Product Lines

Design Patterns

Observer pattern
Intent: Create an event notification of observing objects while modularizing concrete subjects and observers by function and SoLution.

This Presentation
Foundations
Variability-Aware Design Patterns

• Analyzed SPLs for design patterns
• Some design patterns are used to realize variability
• Pattern decomposition on features seems to follow certain rules

Variability-Aware Design Patterns
Role Modeling

- Design patterns are **not** a specific design

- **Role modeling** represents dynamic collaborations as „blueprint“ for a design
  - **Roles** may be mapped to various entities (parts of code, models etc.)
  - **Relations** describe collaborations and restrain possible mapping

  - **Roles may represent patterns**
  - Roles may be mapped to a concrete design in various languages
Challenge

- How to develop SPLs with variability-aware design patterns?
Specification of Design Patterns
Catalog of Variability-Aware Design Patterns

- Cataloged variability-aware design patterns
  - Name
  - Intent
  - ...
  - Design pattern role model (DPRM)

**Usage within SPLs?**
- Family role model (FRM)

```
F5 → F3
(F2 ↔ F6) ∧ (F4 → F6 ∧ F5)
```
Family Role Model (FRM) to capture Variability-Awareness

- Represent demands on configuration options of the feature model (*semantics*)
- Independent of the concrete structure of the feature model (*syntax*)

![Family Role Model Diagrams](image)
Making Design Patterns Variability-Aware

Family Role Model (FRM)
- Captures demands on configuration options of feature model

Mapping FRM to DPRM
- Captures how a design pattern is used within an SPL
- Variability-aware design pattern

Design Pattern Role Model (DPRM)
- Captures entities of the design pattern and their relation
Model-Based Catalog of Variability-Aware Design Patterns
Summary: Design Pattern Specification

FRM to DPRM Mapping

Software Product Line (Problem Space)

Design Pattern Specification

Software Product Line (Solution Space)
Application of Design Patterns
Mapping Family Role Model (FRM) to Concrete Feature Model

- **Figure a)**: F5 → F3
- **Figure b)**: F2
- **Figure c)**: (F2 ↔ F6) ∧ (F4 → F6 ∧ F5)
- **Figure d)**: F4 → ¬F2

Diagram shows the transformation from FRM to feature model with relationships between concepts A, B, C, and D.
Summary: Mapping FRMs to Feature Models

2. Software Product Line (Problem Space)

F1
F2 B F3 F6 A
F4 C F5 D
(F2 ↔ F6) ∧ (F4 → F6 ∧ F5)

Feature Model

1. Design Pattern Specification

Family Role Model (FRM)
Observer ← A
ConcreteObserver
Subject
Observable
Observer ← B
ConcreteObserver
Subject
Observable

FRM to DPRM Mapping

3. Software Product Line (Solution Space)

?
Mapping DPRM to Realization Artifacts

- Multiple different types of realization artifact
  - Java, C++, UML Class Diagrams etc.
  - Pattern implemented differently, due to ...
    - ... language
    - ... variability realization mechanism?

```java
public interface Listener<T> {
    public void update(T object);
}

public abstract class AbstractAudioController implements AudioController {
    protected List<Listener<AudioController>> playListeners;
    public void addPlayListener(Listener<AudioController> l) {
        this.playListeners.add(l);
    }
    protected void notifyPlayListeners() {
        for (Listener<AudioController> l : this.playListeners) {
            l.update(this);
        }
    }
}

public class Mp3Controller extends AbstractAudioController {
    public void play() {
        // ...
        this.notifyPlayListeners();
    }
}

public class PlayerBar {
    // ...
    class PlayListener implements Listener<Controller> {
        public void update(AudioController a) {
            playButton.setEnabled(false);
            pauseButton.setEnabled(true);
            stopButton.setEnabled(true);
        }
    }
}
Generation of Design Pattern Realization Artifacts

- Pattern implemented differently, due to ...
  - ... language
  - ... variability realization mechanism

```java
public class Mp3Controller extends AbstractAudioController {
    // ifdef MP3
    public void play() {
        // ...
        this.notifyPlayListeners();
    }
    // endif
}

Annotated Java (MP3Controller.java)
```

```java
public class Mp3Controller extends AbstractAudioController {
    public void play() {
        // ...
        this.notifyPlayListeners();
    }
}

Feature Module (MP3Controller.java)
```

```java
delta MP3 {
    adds {
        // ...
        public class Mp3Controller extends AbstractAudioController {
            public void play() {
                // ...
                this.notifyPlayListeners();
            }
        }
    }
}

Delta Module (MP3.delta)
```
Model-Based Generation

```java
package org.example2;

import java.util.List;

public class Subject implements Observable {
    private List<Observer> observers = new ArrayList<Observer>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }
}
```
Composers: Eclipse Extension for Model-Based Generation

Composite
Strategy
Observer
Composer 1
Composer 2
Composer 3
Antenna
DeltaJ
FOP
Java
UML Class Diagrams
Summary: Generation of Design Pattern Realization Artifacts

1. Design Pattern Specification
   - Family Role Model (FRM)
   - Design Pattern Role Model (DPRM)
   - FRM to DPRM Mapping

2. Software Product Line (Problem Space)
   - Feature Model
   - (F2 ↔ F6) ∧ (F4 → F6 ∧ F5)

3. Software Product Line (Solution Space)
   - Source Code (Java, C++ etc.)
   - Design Models (UML Class Diagrams)
   - Realization Artifacts
**Implementation**

- Eclipse IDE
- Model-Based with EMF Ecore
  - Meta model for Role Models (FRM/DPRM), Design Pattern Catalog, Feature Model
  - Generation is Model-Based (but does not have to be)

Mockup!
## Currently Supported Pattern Realizations

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Variability Realization Mechanism</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observer</td>
<td>Antenna</td>
<td>Java</td>
</tr>
<tr>
<td></td>
<td>Feature-Oriented Programming</td>
<td>Java, UML Class Diagrams</td>
</tr>
<tr>
<td></td>
<td>DeltaJ</td>
<td>Java</td>
</tr>
<tr>
<td>Composite</td>
<td>Antenna</td>
<td>Java</td>
</tr>
<tr>
<td></td>
<td>Feature-Oriented Programming</td>
<td>Java, UML Class Diagrams</td>
</tr>
<tr>
<td></td>
<td>DeltaJ</td>
<td>Java</td>
</tr>
<tr>
<td>Strategy</td>
<td>Antenna</td>
<td>Java</td>
</tr>
<tr>
<td></td>
<td>Feature-Oriented Programming</td>
<td>Java, UML Class Diagrams</td>
</tr>
<tr>
<td></td>
<td>DeltaJ</td>
<td>Java</td>
</tr>
</tbody>
</table>
Conclusion

- Generative Software Product Line Development using Variability-Aware Design Patterns
- Support proactive and reactive development