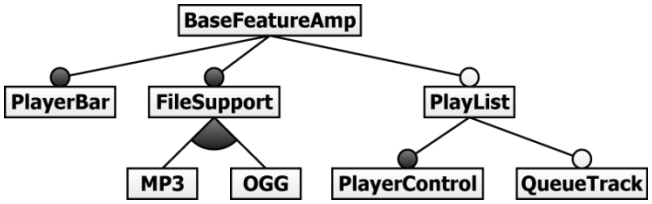
Generative Software Product Line Development using Variability-Aware Design Patterns

Christoph Seidl, Sven Schuster, Ina Schaefer

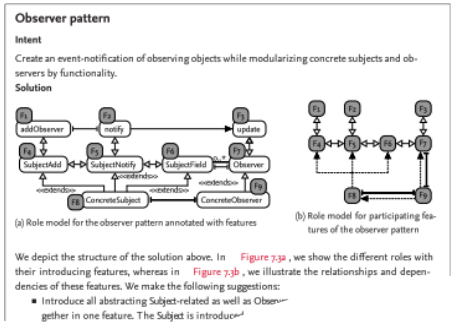
May 16, 2015

Idea: Use Best-Practices for Software Product Line Design

Sven Schuster‘s Presentation



Software Product Lines


Observer pattern

Intent

Create an event-notification of observing objects while modularizing concrete subjects and observers by functionality.

Solution

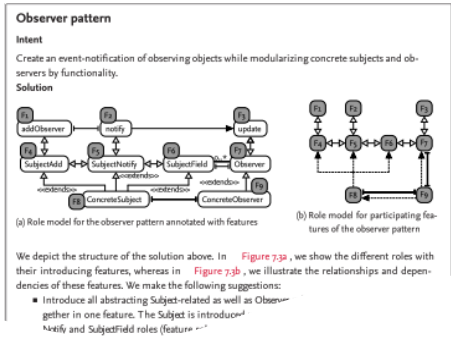
(a) Role model for the observer pattern annotated with features

(b) Role model for participating features of the observer pattern

We depict the structure of the solution above. In Figure 7.3a, we show the different roles with their introducing features, whereas in Figure 7.3b, we illustrate the relationships and dependencies of these features. We make the following suggestions:

- Introduce all abstracting Subject-related as well as Observer together in one feature. The Subject is introduced

Design Patterns


Observer pattern

Intent

Create an event-notification of observing objects while modularizing concrete subjects and observers by functionality.

Solution

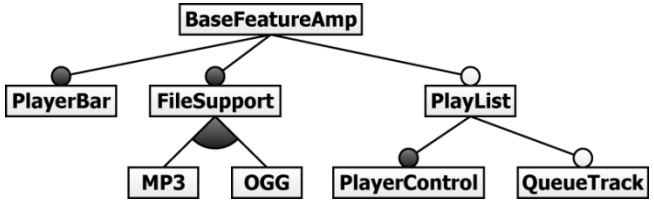
(a) Role model for the observer pattern annotated with features

(b) Role model for participating features of the observer pattern

We depict the structure of the solution above. In Figure 7.3a, we show the different roles with their introducing features, whereas in Figure 7.3b, we illustrate the relationships and dependencies of these features. We make the following suggestions:

- Introduce all abstracting Subject-related as well as Observer together in one feature. The Subject is introduced

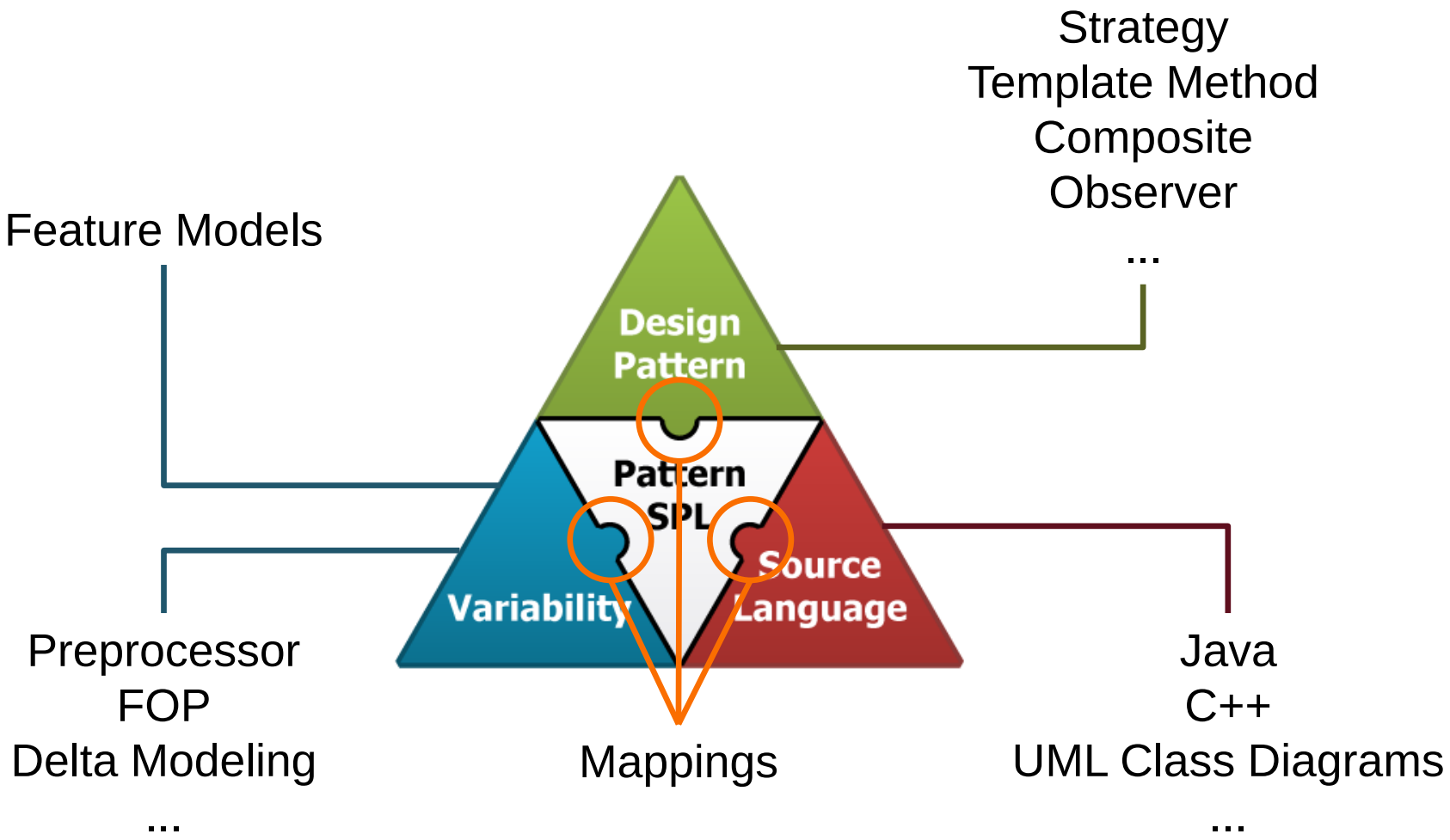
Design Patterns



Software Product Lines

This Presentation

Overview

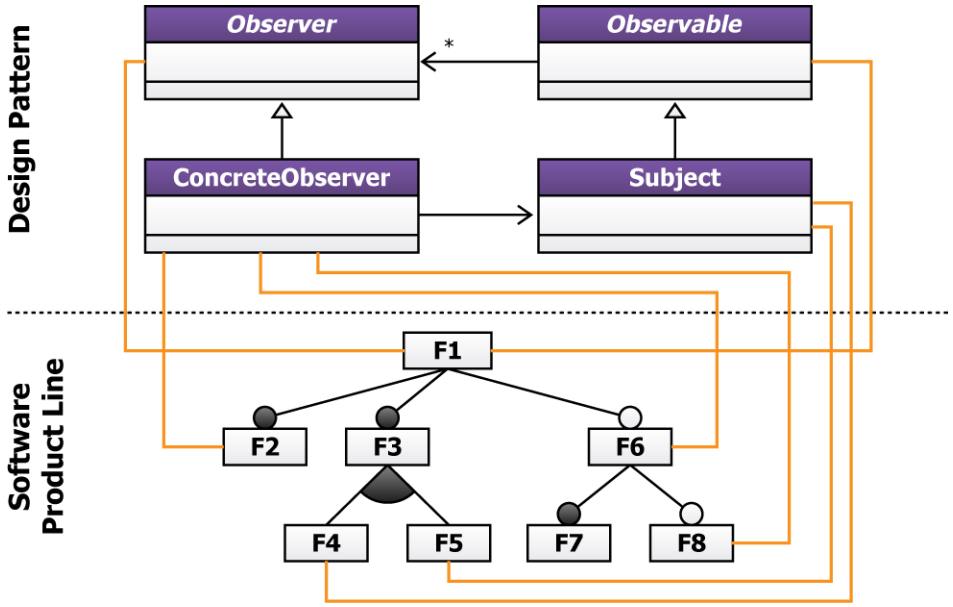

Strategy
Template Method
Composite
Observer
...
Feature Models
Design Pattern
Pattern SPL
Variability
Source Language
Preprocessor
FOP
Delta Modeling
...
Mappings
Java
C++
UML Class Diagrams
...

Technische Universität Braunschweig

Foundations

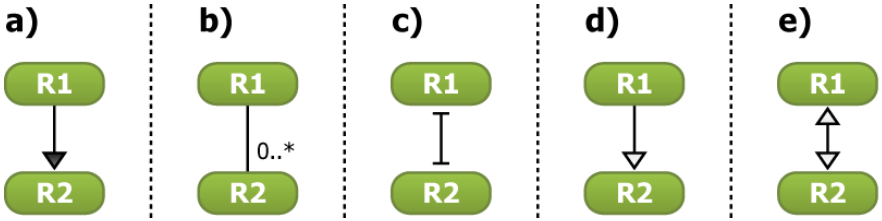
Variability-Aware Design Patterns

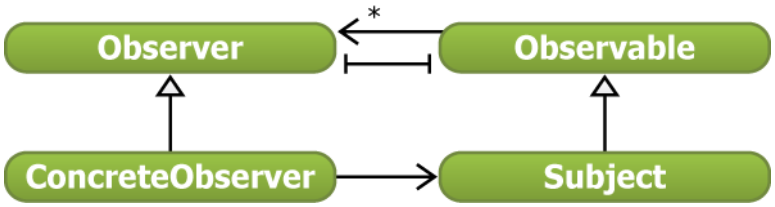
- Analyzed SPLs for design patterns
- Some design patterns are used to realize variability
- Pattern decomposition on features seems to follow certain rules
- ➢ **Variability-Aware Design Patterns**



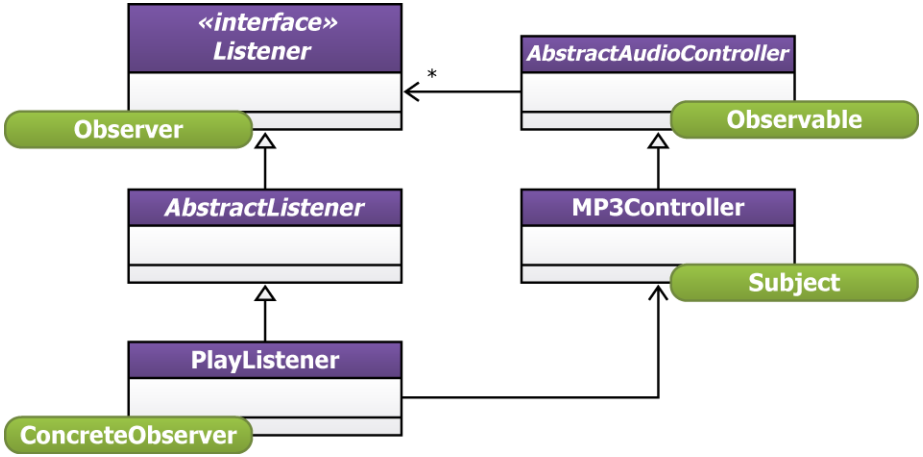
Role Modeling

- 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

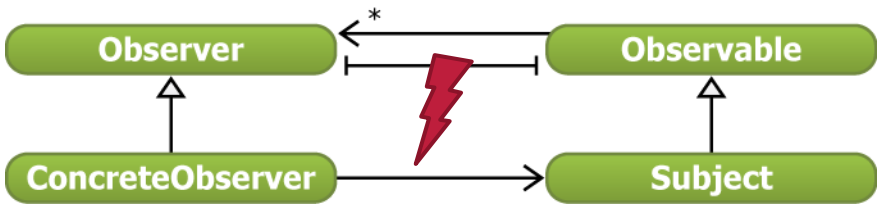
- How to *develop* SPLs with variability-aware design patterns?

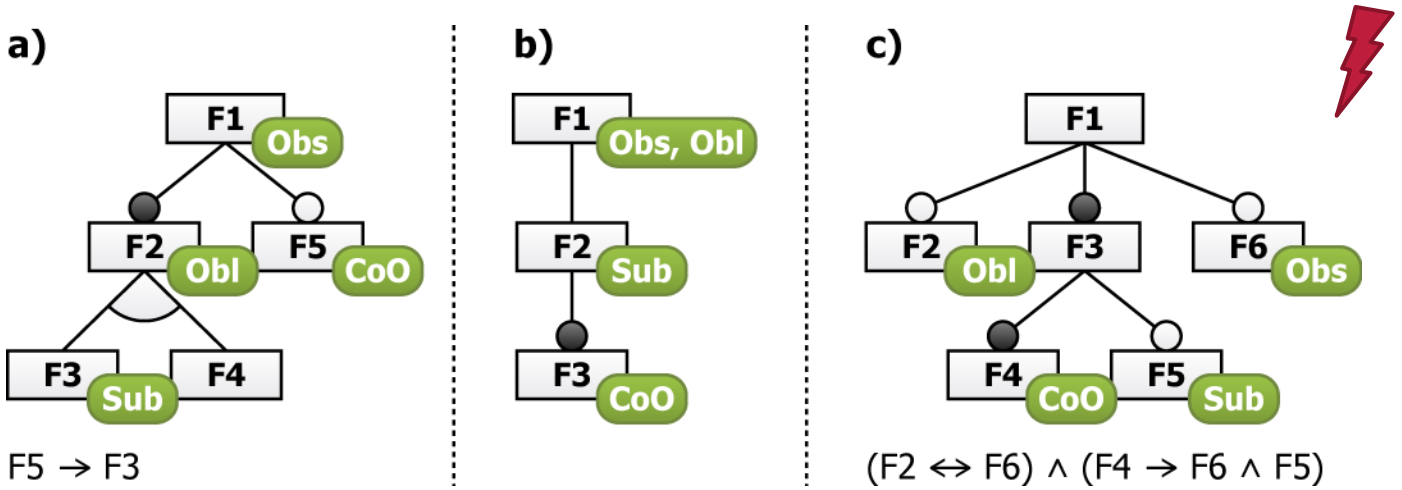
Technische Universität Braunschweig

Specification of Design Patterns

Catalog of Variability-Aware Design Patterns

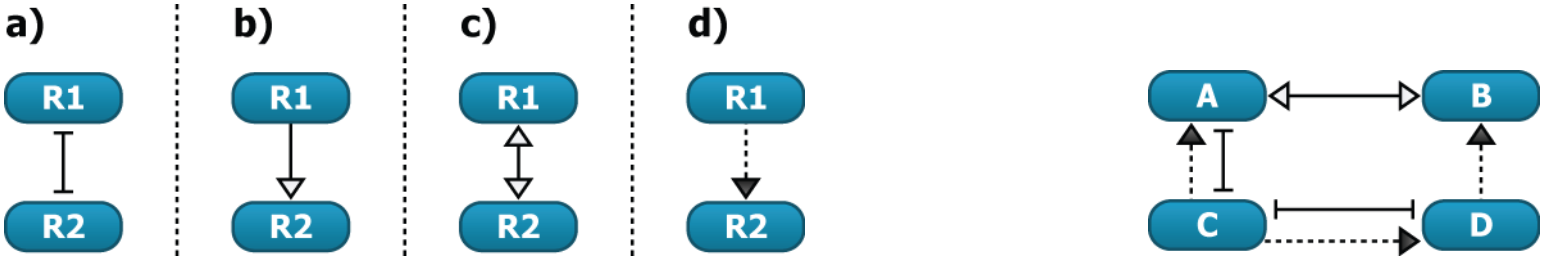
- Cataloged variability-aware design patterns
  - Name
  - Intent
  - ...
  - Design pattern role model (DPRM)
  - **Usage within SPLs?**
    - ➢ **Family role model (FRM)**

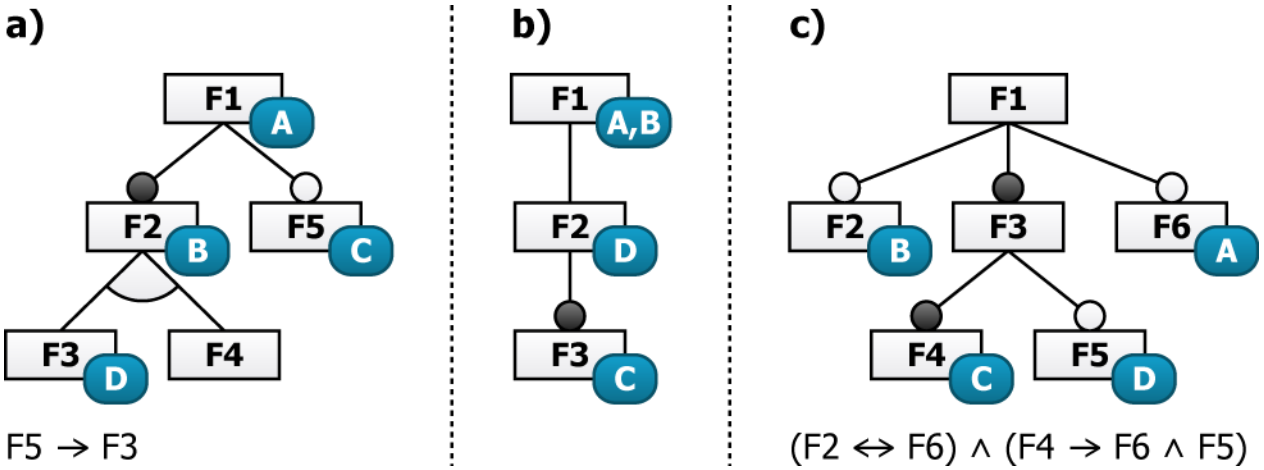




Family Role Model (FRM) to capture Variability-Awareness

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

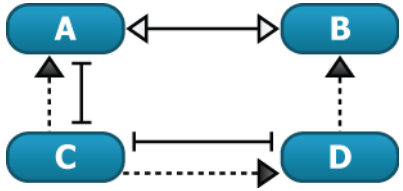




Making Design Patterns Variability-Aware

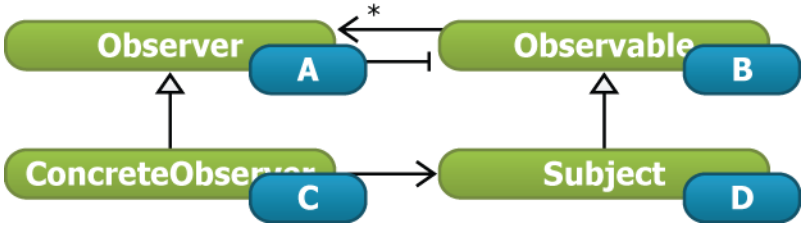
Family Role Model (FRM)

- Captures demands on configuration options of feature model



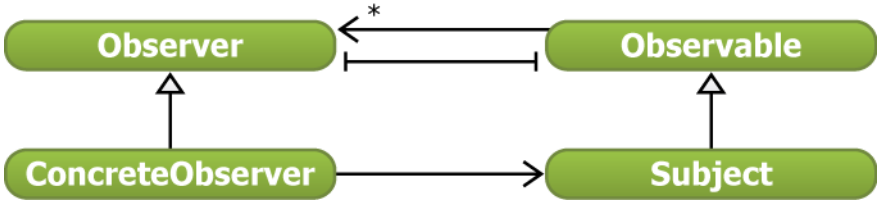
Mapping FRM to DPRM

- Captures how a design pattern is used within an SPL
- **Variability-aware design pattern**

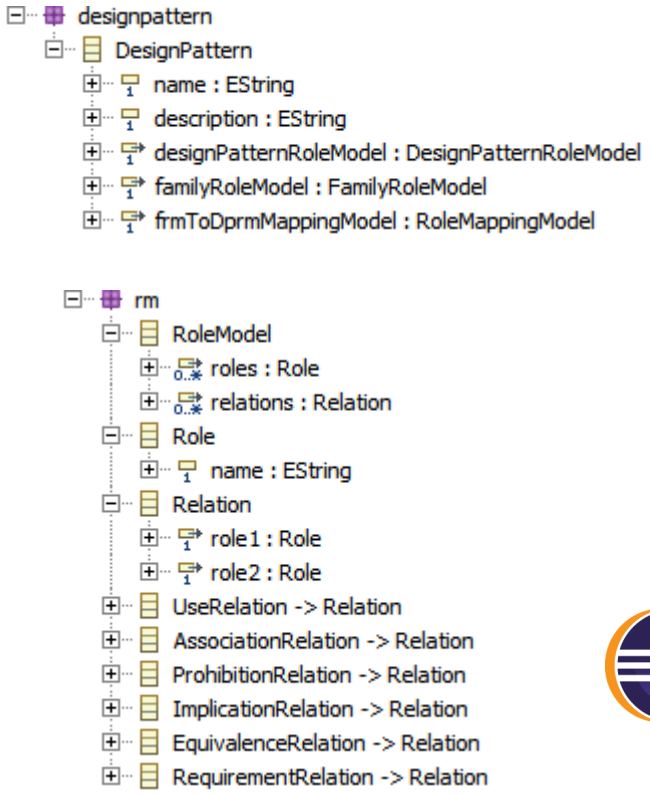


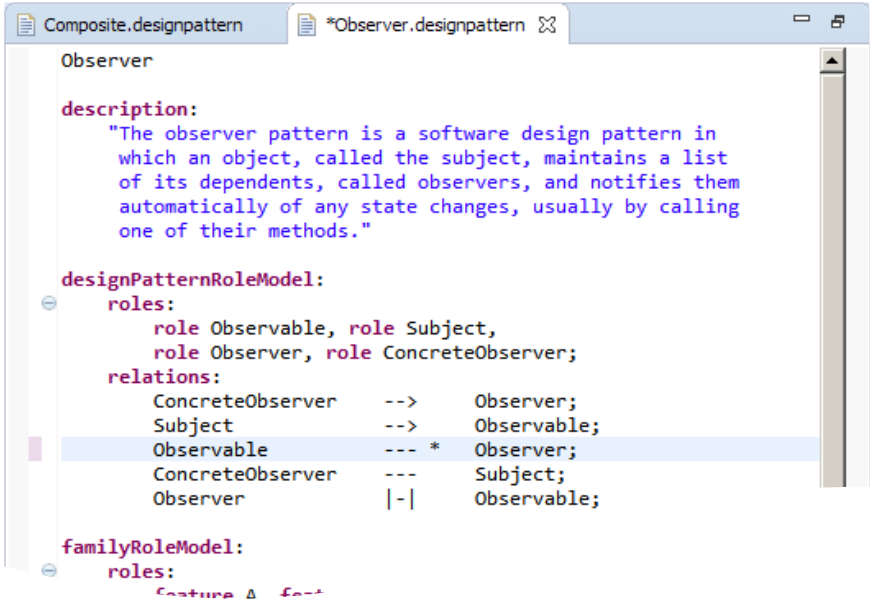
Design Pattern Role Model (DPRM)

- Captures entities of the design pattern and their relation

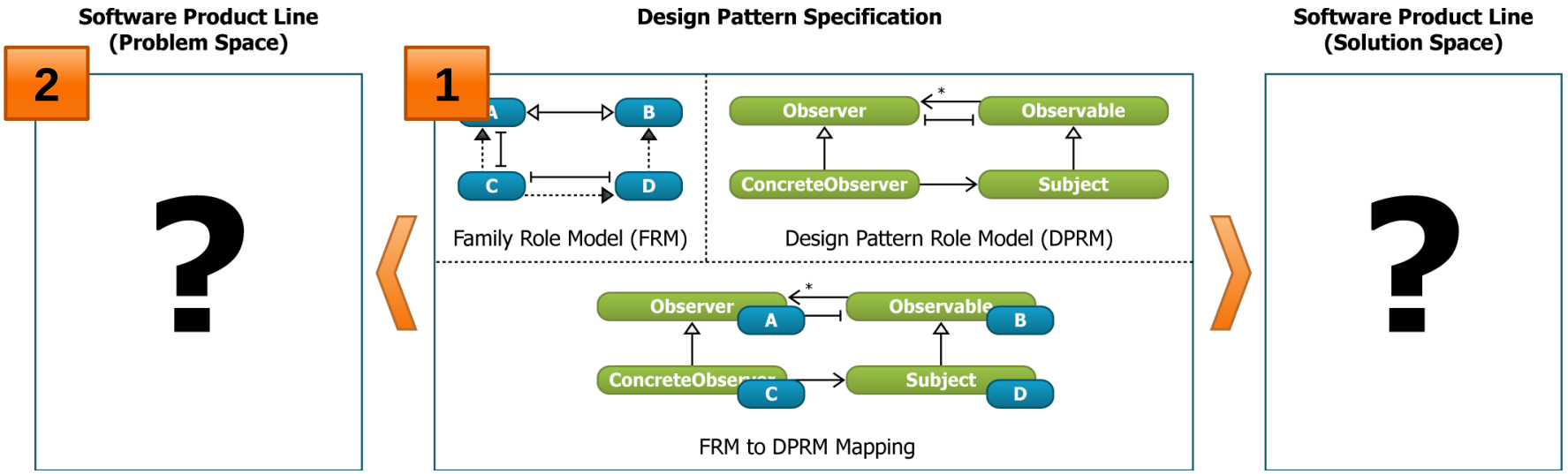


Model-Based Catalog of Variability-Aware Design Patterns



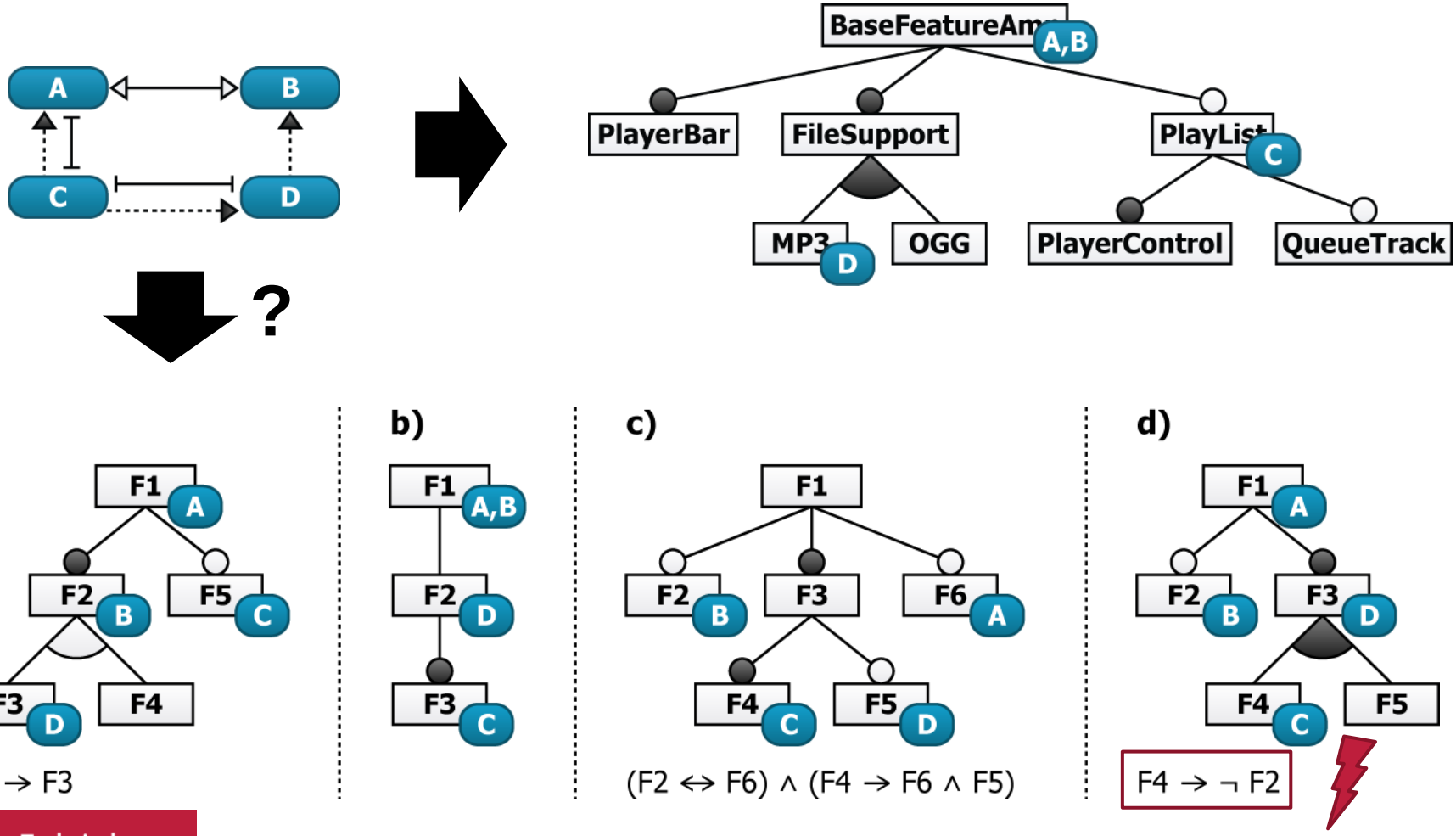


Summary: Design Pattern Specification

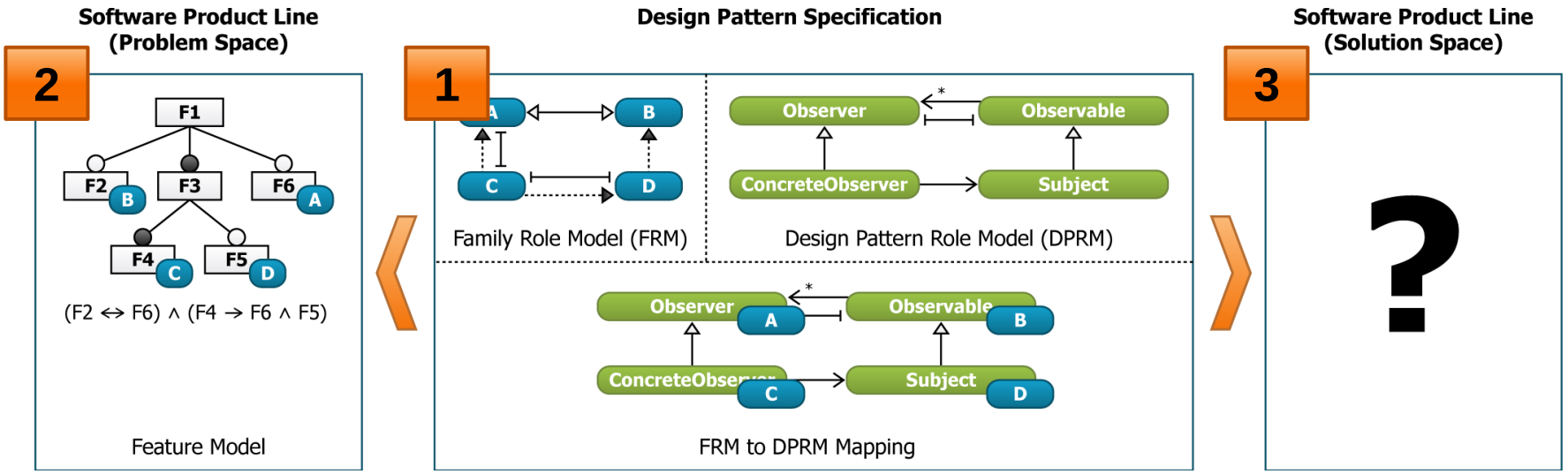


Application of Design Patterns

Mapping Family Role Model (FRM) to Concrete Feature Model

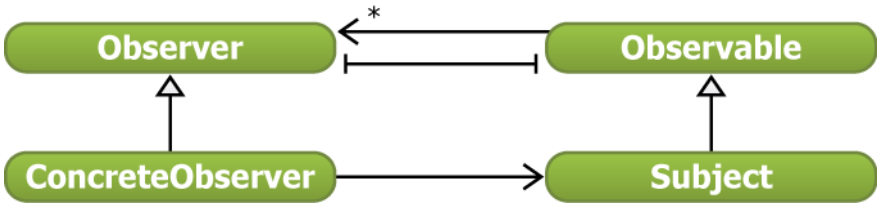


Summary: Mapping FRMs to Feature Models



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?



Observer

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

Observable

```
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);
    }
  }
}
```

Subject

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

ConcreteObserver

```
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**

a)

D

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

Annotated Java (MP3Controller.java)

b)

D

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

Feature Module (MP3Controller.java)

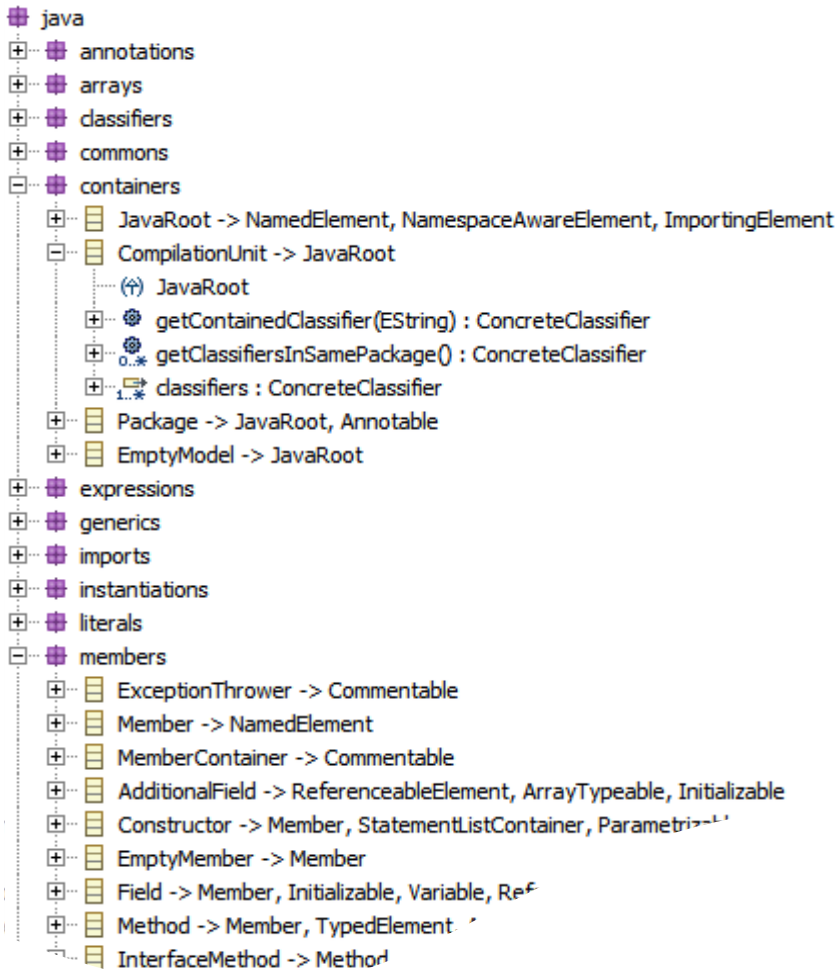
c)

D

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

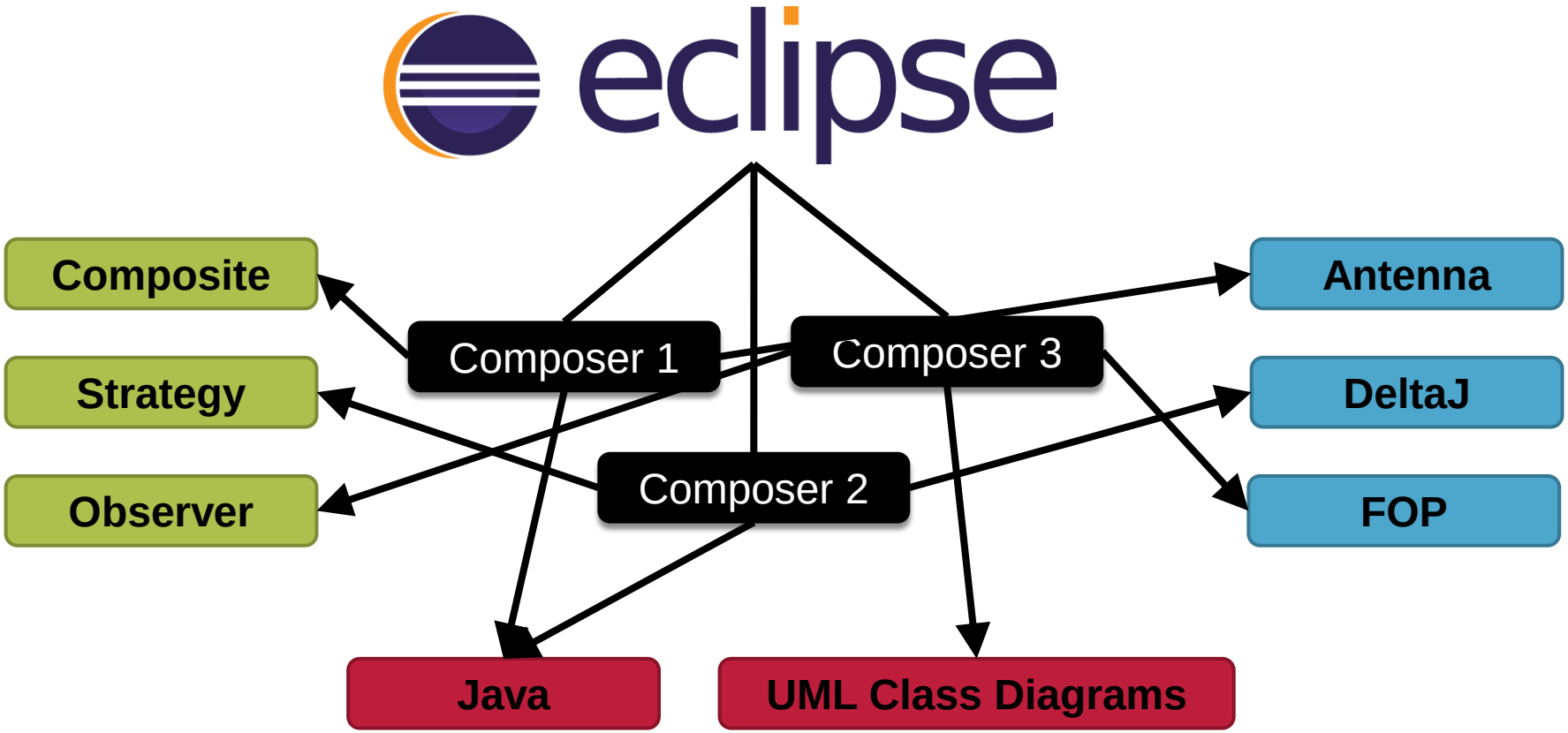
Delta Module (MP3.deltaj)

Model-Based Generation

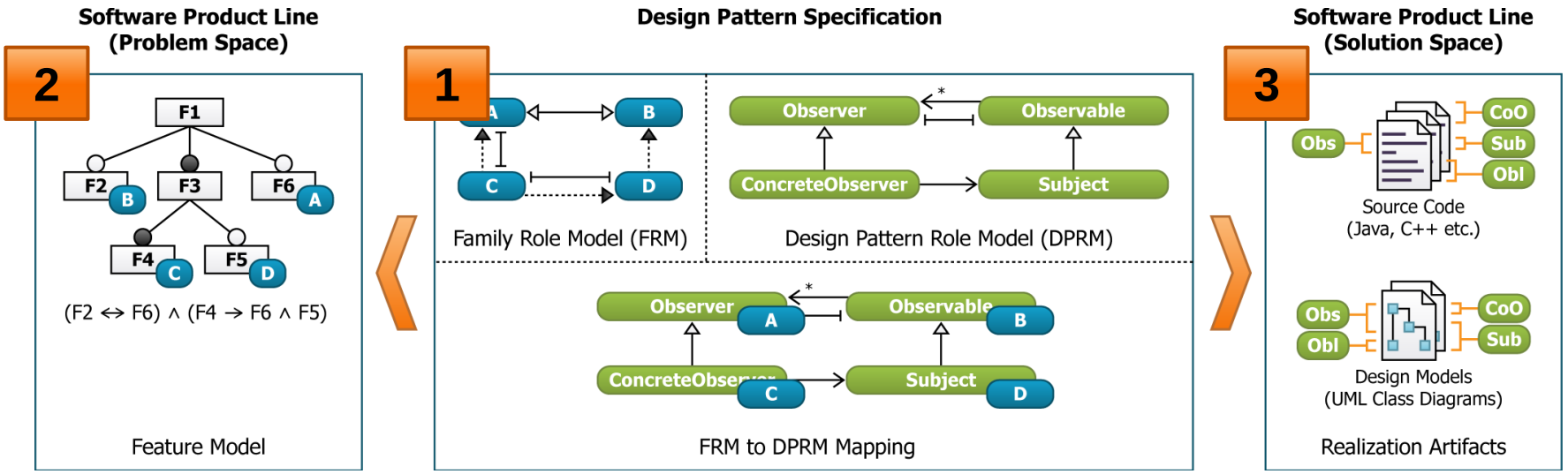



```java
package org.example2;

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

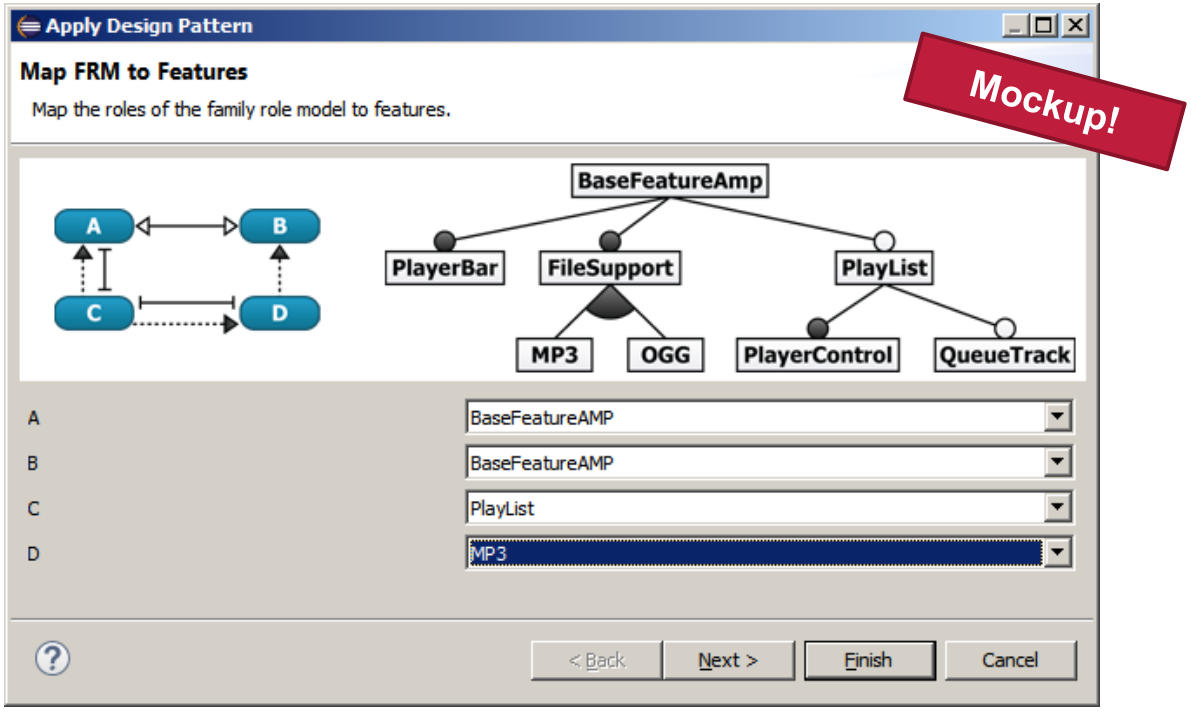


Summary: Generation of Design Pattern Realization Artifacts



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)

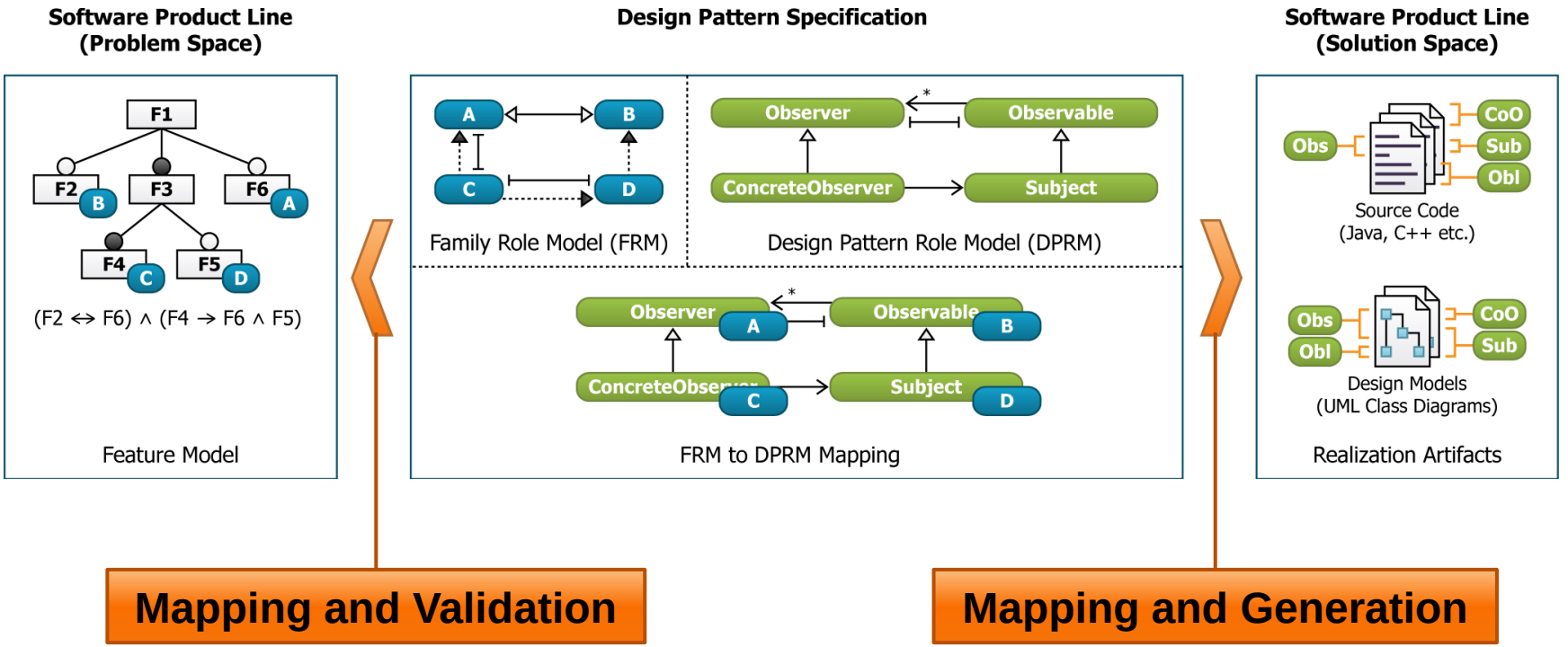


Currently Supported Pattern Realizations

Pattern	Variability Realization Mechanism	Language
Observer	Antenna	Java
	Feature-Oriented Programming	Java, UML Class Diagrams
	DeltaJ	Java
Composite	Antenna	Java
	Feature-Oriented Programming	Java, UML Class Diagrams
	DeltaJ	Java
Strategy	Antenna	Java
	Feature-Oriented Programming	Java, UML Class Diagrams
	DeltaJ	Java

Conclusion

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



Questions? Comments? Feedback?