Variability Models must not be Invariant

Elmar Jürgens, Markus Pizka,
{juergens, pizka}@in.tum.de
Technische Universität München

Outline

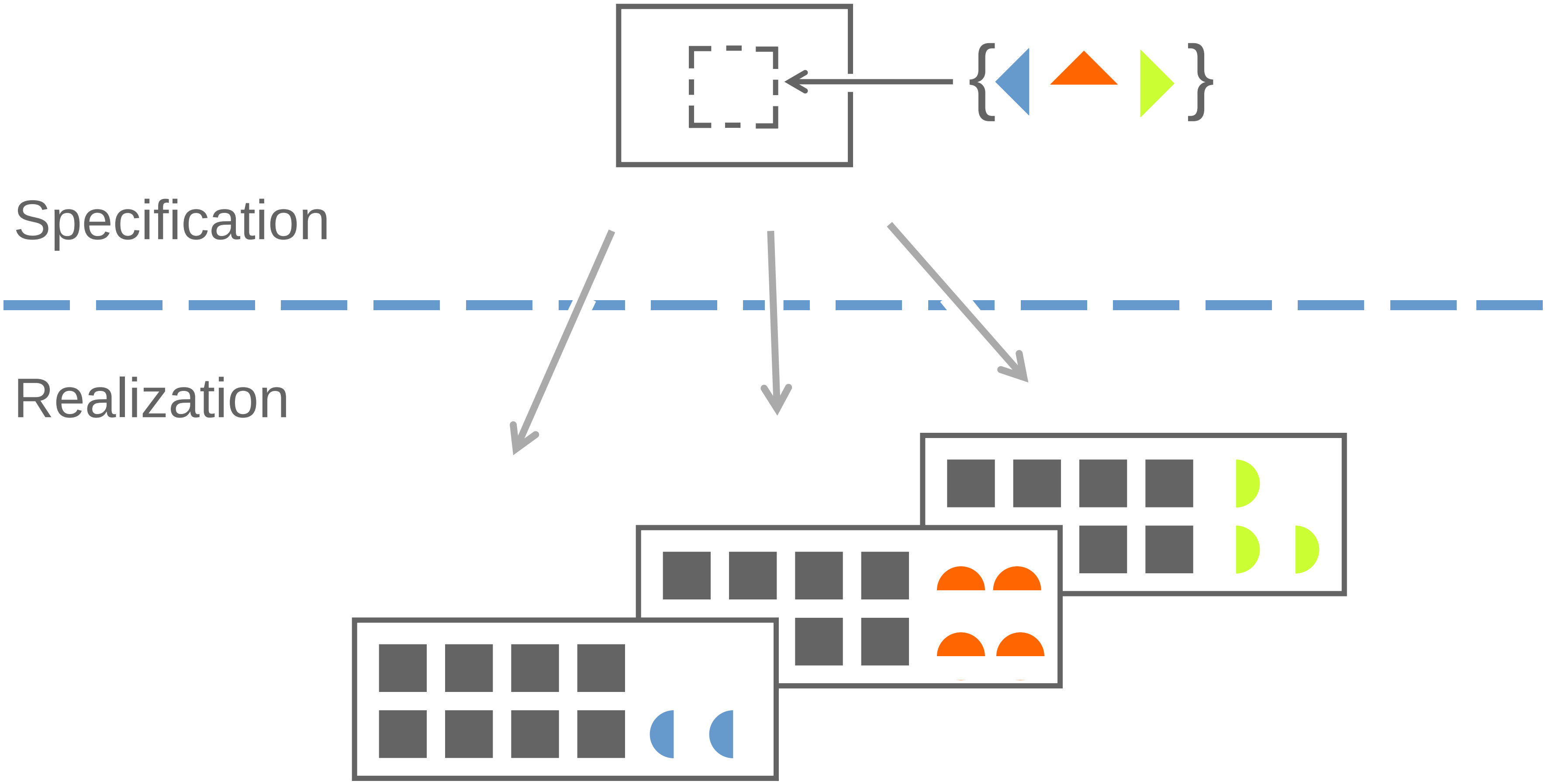
Abstractions and Variability

Invariance and Reuse

Variability Model Evolution

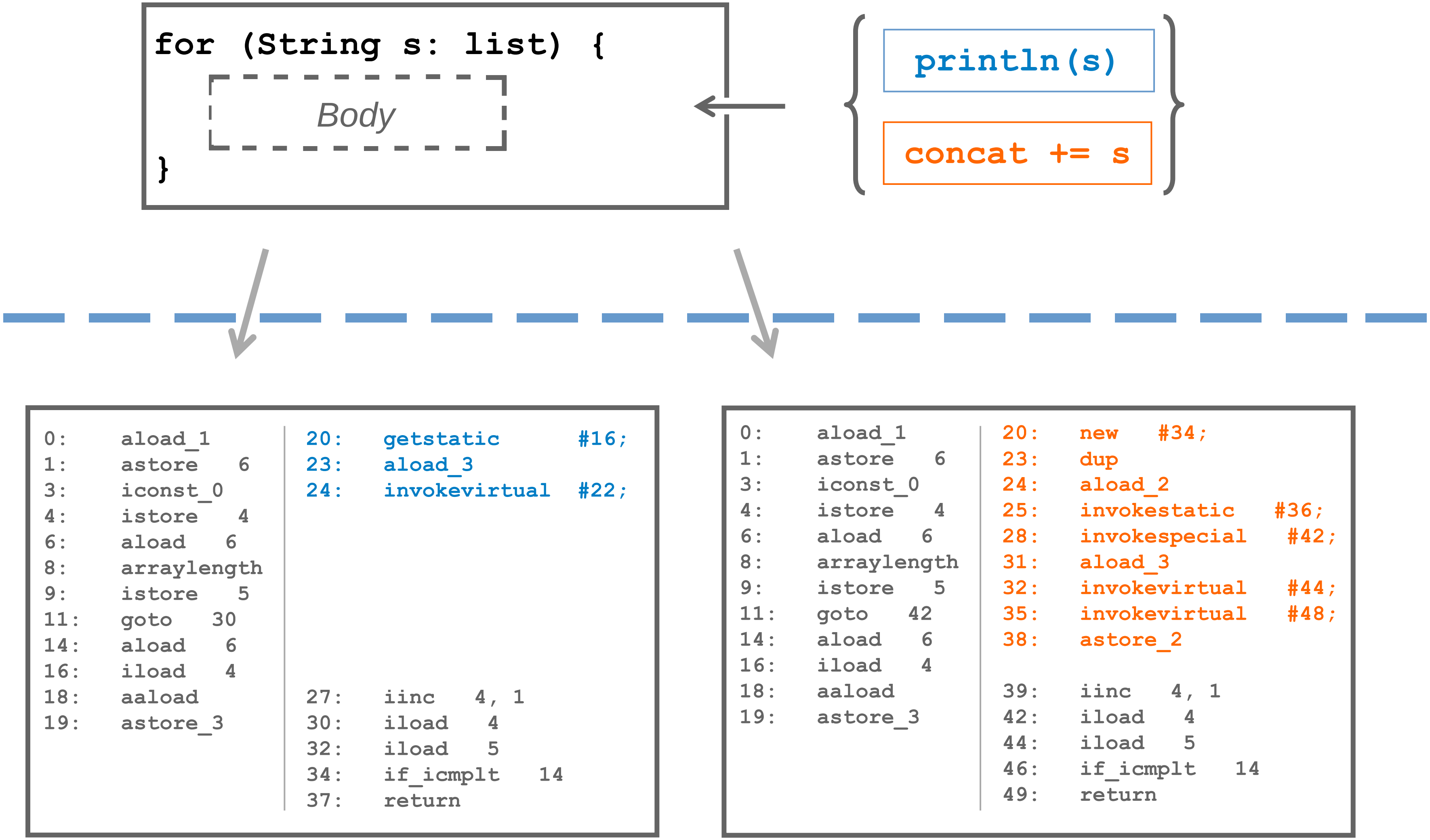
Conclusion

Abstractions and Variability



"Software Reuse", Charles W. Krueger, 1992

Example: GPPL



```
for (String s: list) {
    Body
}
```

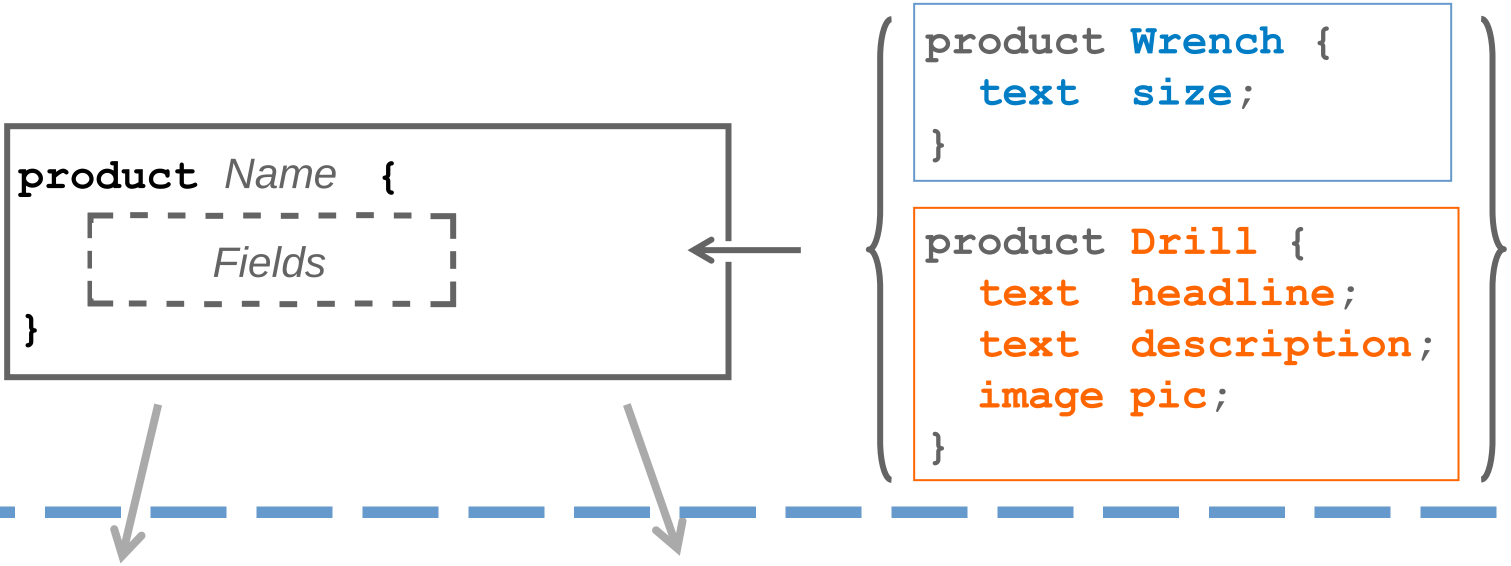
```
println(s)
```

```
concat += s
```

```
0:     aload_1            20:    getstatic       #16;
1:     astore    6        23:    aload_3
3:     iconst_0           24:    invokevirtual   #22;
4:     istore    4
6:     aload    6
8:     arraylength
9:     istore    5
11:    goto    30
14:    aload    6
16:    iload    4
18:    aaload             27:    iinc    4, 1
19:    astore_3           30:    iload    4
                          32:    iload    5
                          34:    if_icmplt    14
                          37:    return
```

```
0:     aload_1            20:    new    #34;
1:     astore    6        23:    dup
3:     iconst_0           24:    aload_2
4:     istore    4        25:    invokestatic     #36;
6:     aload    6         28:    invokespecial    #42;
8:     arraylength        31:    aload_3
9:     istore    5        32:    invokevirtual    #44;
11:    goto    42         35:    invokevirtual    #48;
14:    aload    6         38:    astore_2
16:    iload    4
18:    aaload             39:    iinc    4, 1
19:    astore_3           42:    iload    4
                          44:    iload    5
                          46:    if_icmplt    14
                          49:    return
```

Example: DSL



```
product Name {
   Fields
}
```

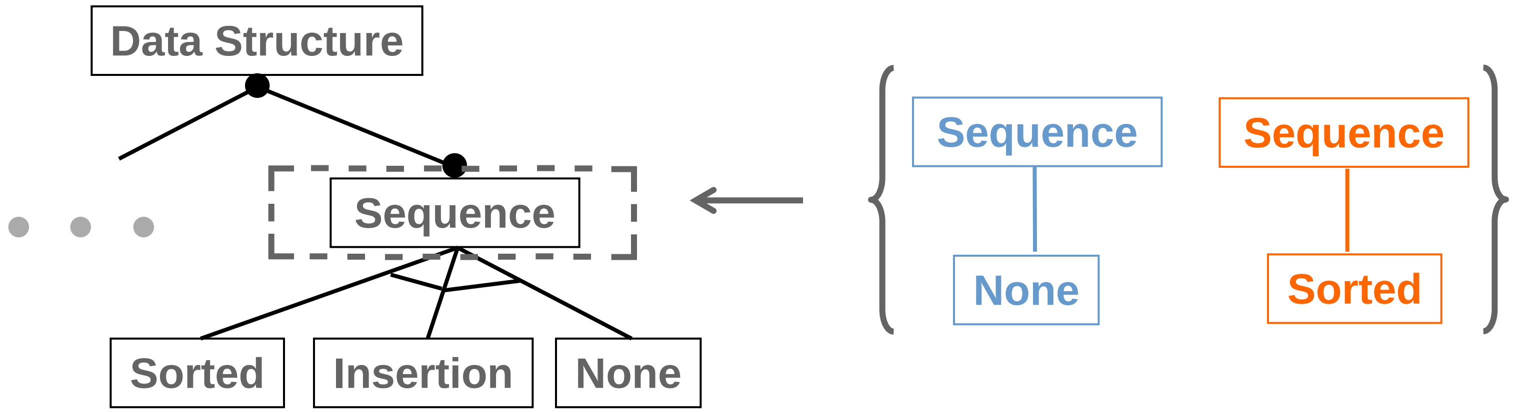
```
product Wrench {
   text  size;
}
```

```
product Drill {
   text  headline;
   text  description;
   image pic;
}
```

```
package view.display.generated;

import model.documents.generated.*;
import view.display.Display;
import view.display.controls.*;

/** Display for Wrench documents */
public class WrenchDisplay extends Display {

   private static final long serialVersionUID = 1L;

   private LabeledSingleLineTextLabel Size;


   @Override
   protected void initControls() {
      Size = new LabeledSingleLineTextLabel();
      Size.setLabel("Size");
      this.add(Size);
   }


   @Override
   public void displayDocument() {
      Wrench doc = (Wrench) document;

      Size.bind(doc.getSize());
   }
}
```

```
package view.display.generated;

import model.documents.generated.*;
import view.display.Display;
import view.display.controls.*;

/** Display for Drill documents */
public class DrillDisplay extends Display {

   private static final long serialVersionUID = 1L;

   private LabeledSingleLineTextLabel Headline;
   private LabeledMultiLineTextLabel  ShortDescription;
   private LabeledImageDisplay ProductImage;


   @Override
   protected void initControls() {
      Headline = new LabeledSingleLineTextLabel();
      Headline.setLabel("Headline");
      this.add(Headline);

      ShortDescription = new LabeledMultiLineTextLabel();
      ShortDescription.setLabel("ShortDescription");
      this.add(ShortDescription);

      ProductImage = new LabeledImageDisplay();
      ProductImage.setLabel("ProductImage");
      this.add(ProductImage);
   }

@Override
   public void displayDocument() {
      Drill doc = (Drill) document;

      Headline.bind(doc.getHeadline());
      ShortDescription.bind(doc.getShortDescription());
      ProductImage.bind(doc.getProductImage());
   }
}
```

Example: Feature Model



• • •

Invariance and Reuse

Abstraction creation aims at maximizing

- Effort saved per application of abstraction
- Number of times abstraction can be applied

Invariant parts determine both the

- Amount of reused artifacts / information

=> Amount of saved effort per use

- Amount of commonality constraints between instances

=> Number of times the abstraction can be applied

Deciding what is variable (and thus also what is invariant) determines reuse benefit of an abstraction!

Invariance Dilemma

Increasing Invariance

- Increases saved effort per abstraction application
- Reduces number of times it can potentially be applied

=> Conflicting Goals!

Optimal Amount of Invariance

- As much invariance as the abstraction use cases allow for

=> All use cases of the abstraction must be known

=> Not possible in practice, since future use cases are unknown!

Abstraction creation is quest for lesser evil

- Loss of potential productivity gain
- Loss of potential abstraction use cases

Avoiding the Dilemma

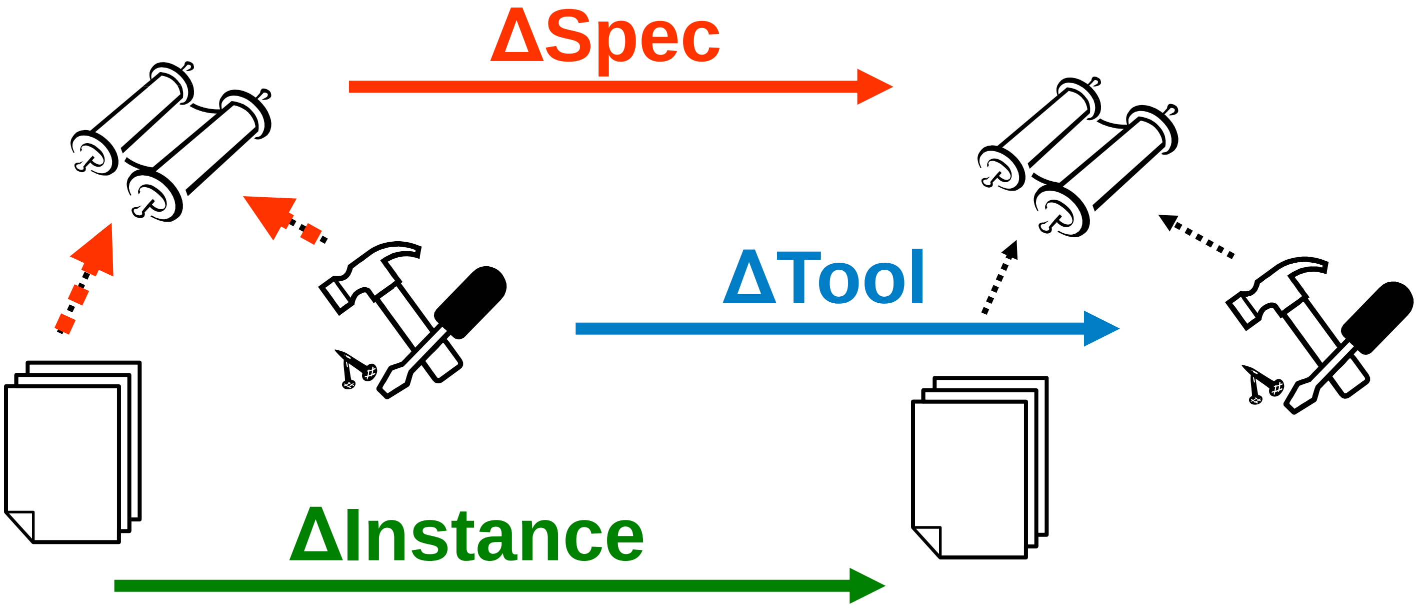
- Only consider currently known use cases

  (=> ignore uncertain future uses cases)
- Make all commonalities between use cases invariants of the abstraction

  => Optimal abstraction reuse benefit
- Evolve partition between variability and invariance as new (uncovered) use cases arise

Invariance Dilemma can be avoided if the partition between variable and invariant information can change over time.

Variability Model Evolution



- Infeasible if done manually
- Compensational effort must be automated to a high degree

(Some) Existing Approaches

Schema Evolution

Adapt Data to changes to DB-schema

Grammar Engineering

Systematic grammar development, co-evolution of grammars and words

Refactoring

Adaptation of use sites to changes to definitions of methods, classes, …

Feature Model Synchronization

Adaptation of configurations to changes to their feature models

Language Evolution

Adaptation of words and processing tools to changes to language spec.

Discussion

Same fundamental problem

- All approaches try to automate the compensational changes to instances after changes to their specification
- Focus on different formalisms for abstraction specification notation

Limitations

- Several approaches are still rather young
- Limited automation for operations beyond refactoring
- Mostly ignore processing tools completely

Conclusion

- Necessity to evolve variability models
- Evolution is difficult, difficulty is essential, not accidental
- General problem, not limited to variability modeling

- Existing approaches limited to single abstraction formalism; fail to tackle problem in generality
- Existing approaches mainly target instance migration, largely neglect adaptation of semantics, tools, …

- Hope: Unified solution attempt could deliver results or insights relevant across abstraction formalisms.