Tool Supported (Multi-Level) Language Evolution

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

Outline

Variability, Abstractions, DSLs

Case Study: DSL for product catalog CMS

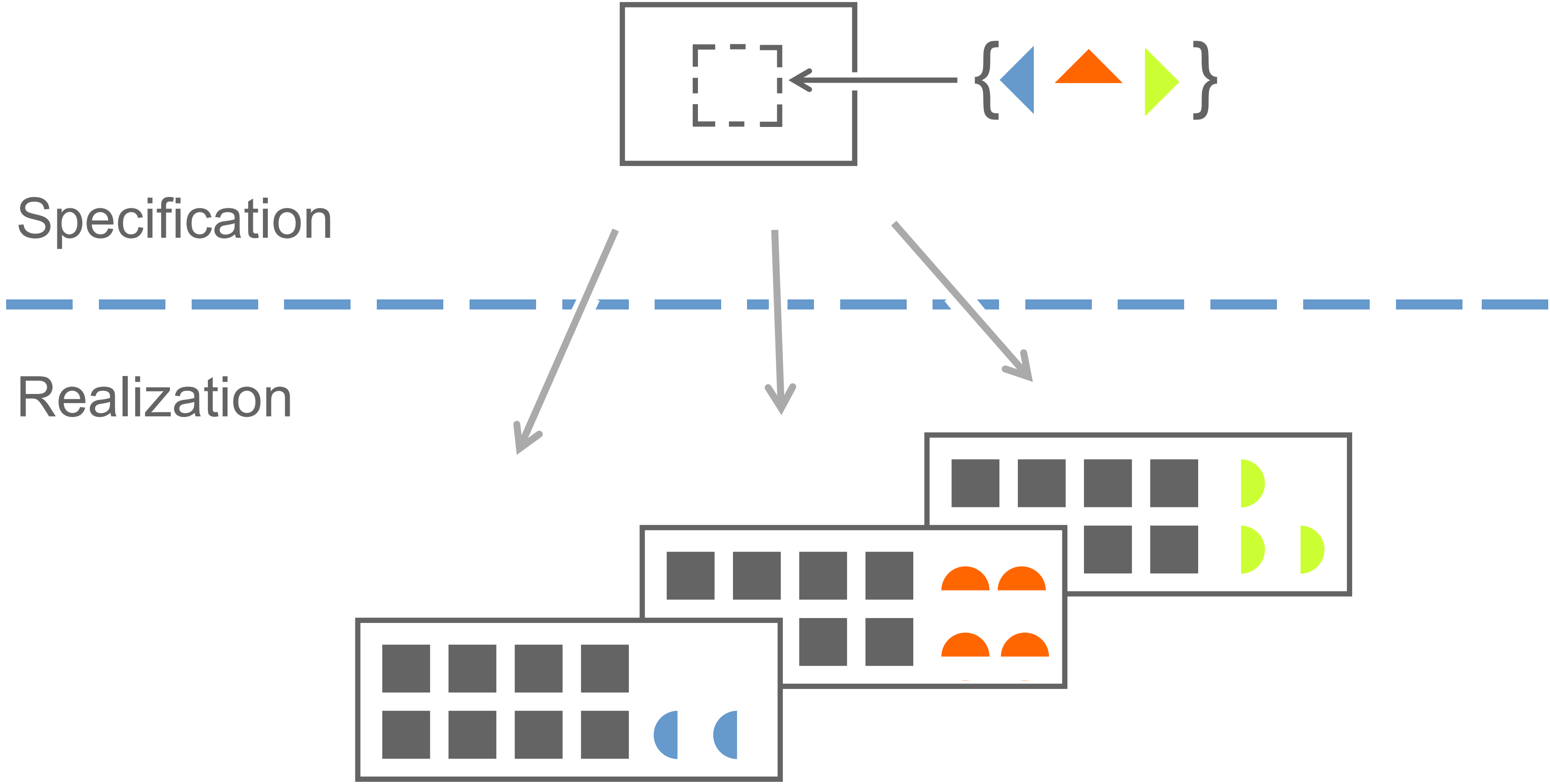
Tool Approach

Limitations & Outlook

Claim

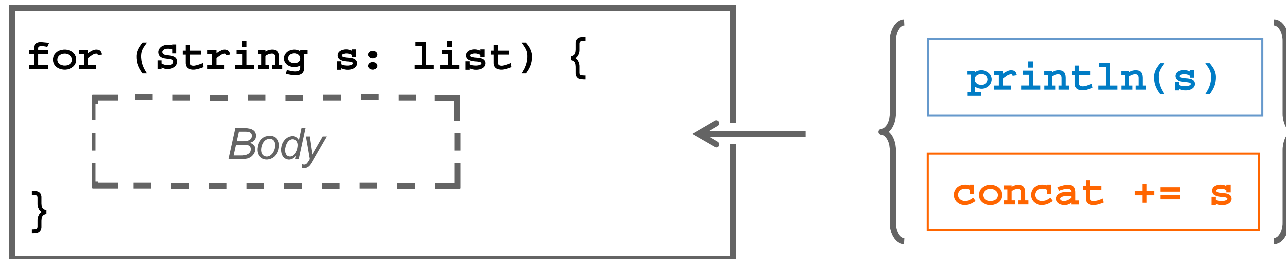
Every abstraction mechanism deals with commonality and variability.

Abstractions and Variability



”Software Reuse”, Charles W. Krueger, 1992

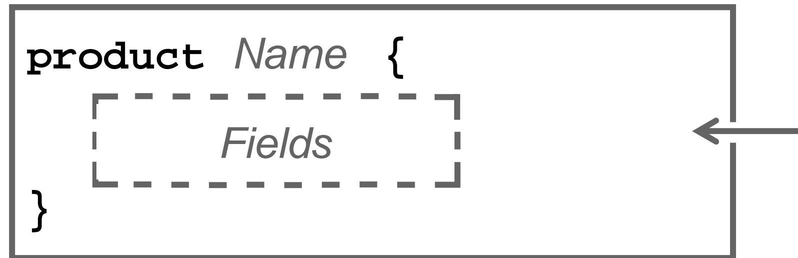
Example: GPPL




```
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 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());
   }
}
```

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

Claim

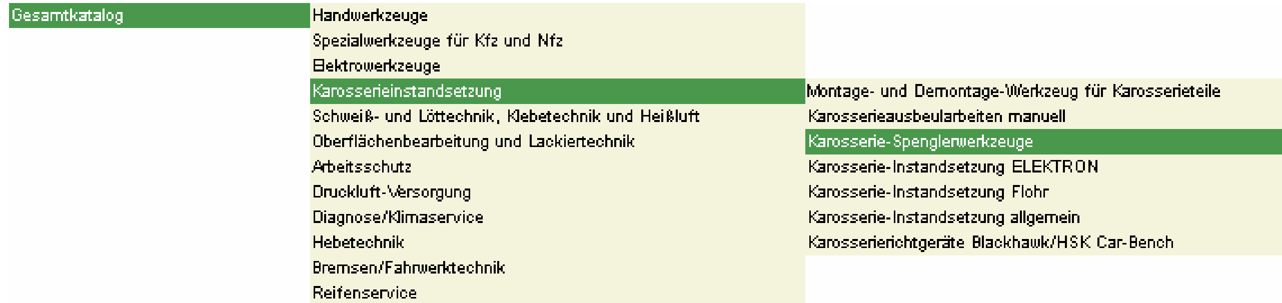
The Invariance Dilemma can be avoided by bottom-up abstraction development.

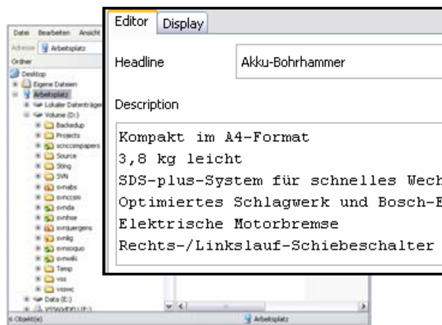
Avoiding the Dilemma

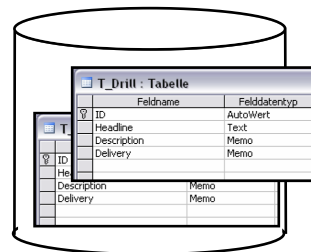
- Only consider currently known use cases
  (=> ignore uncertain future uses cases)
- Make all commonalities between use cases invariants of the abstraction
  => Optimal abstraction reuse benefit
- Evolve partition between variability and invariance as new (uncovered) use cases arise

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

CS: Product Catalog Description Language







Catalog Description Language (2)

```
catalog {
    document Drill {
        field SinglelineText Headline;
        field MultiLineText Description;
        field MultiLineText Delivery;
    }
}
```

Editor | Display

Headline Akku-Bohrhammer

Description

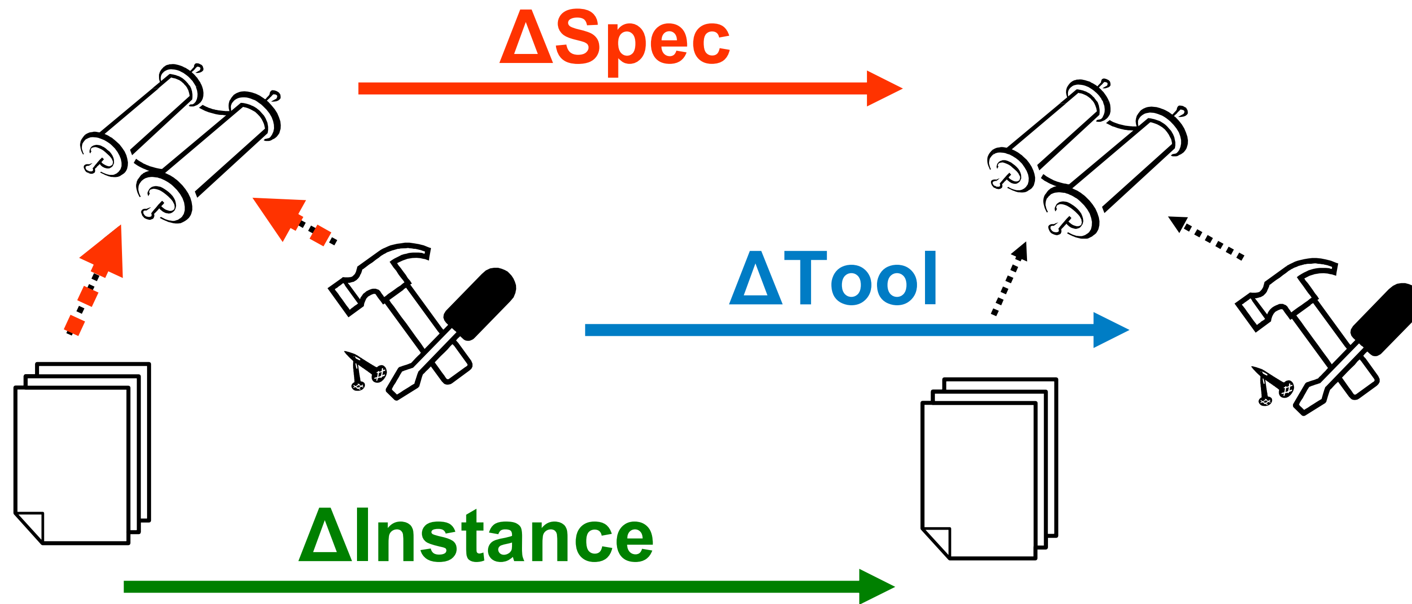
Kompakt im A4-Format
3,8 kg leicht
SDS-plus-System für
Optimiertes Schlag
Elektrische Motorb
Rechts-/Linkslauf-Schiebeschalter mit Einschaltsperre

T_Drill : Tabelle

Feldname	Felddatentyp
ID	AutoWert
Headline	Text
Description	Memo
Delivery	Memo

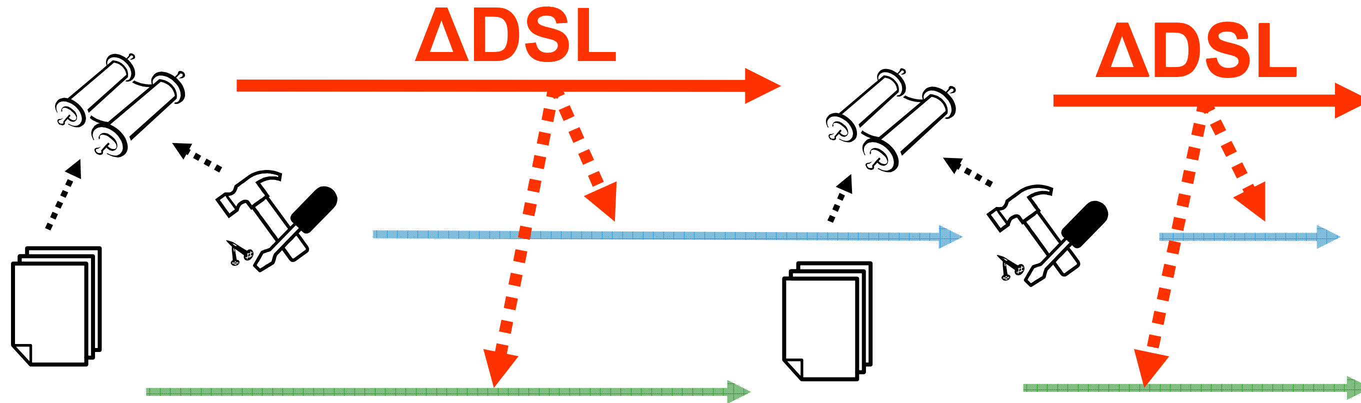
Problems with language evolution …

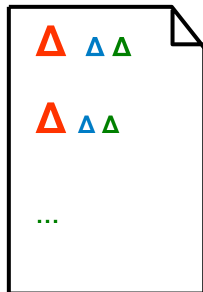
Language Evolution



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

Approach

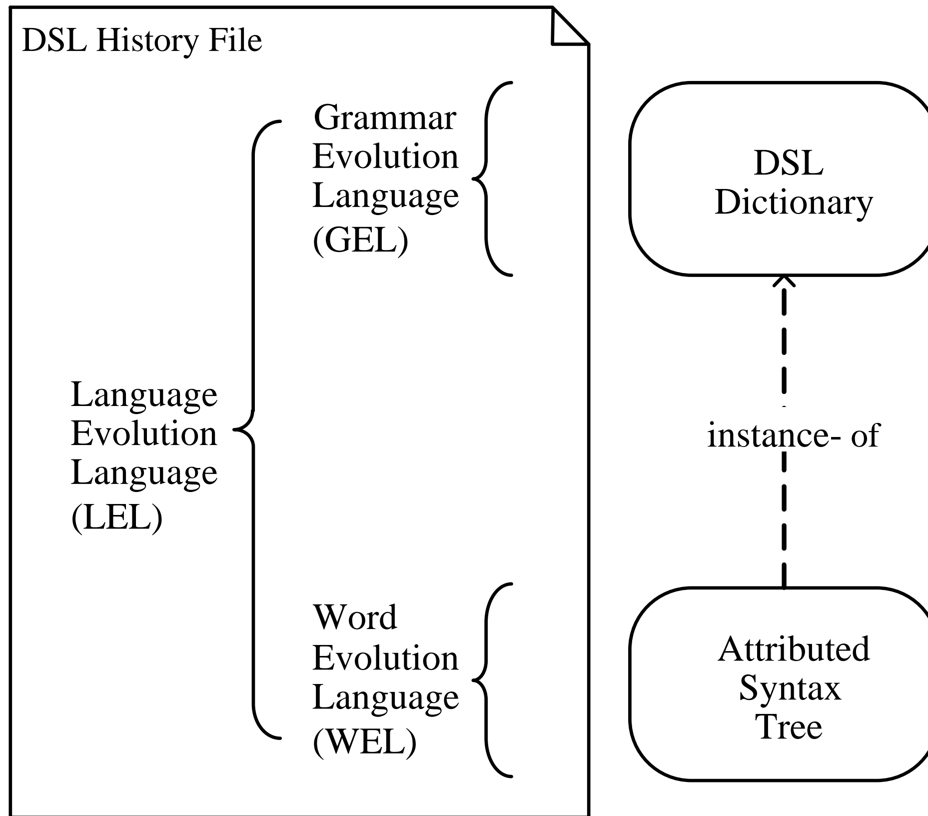




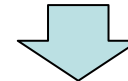
DSL History

- First version of a DSL
- Deltas btw. consecutive versions

DSL History

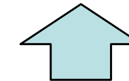


GEL
- DSL Dictionary manipulation
- Complete



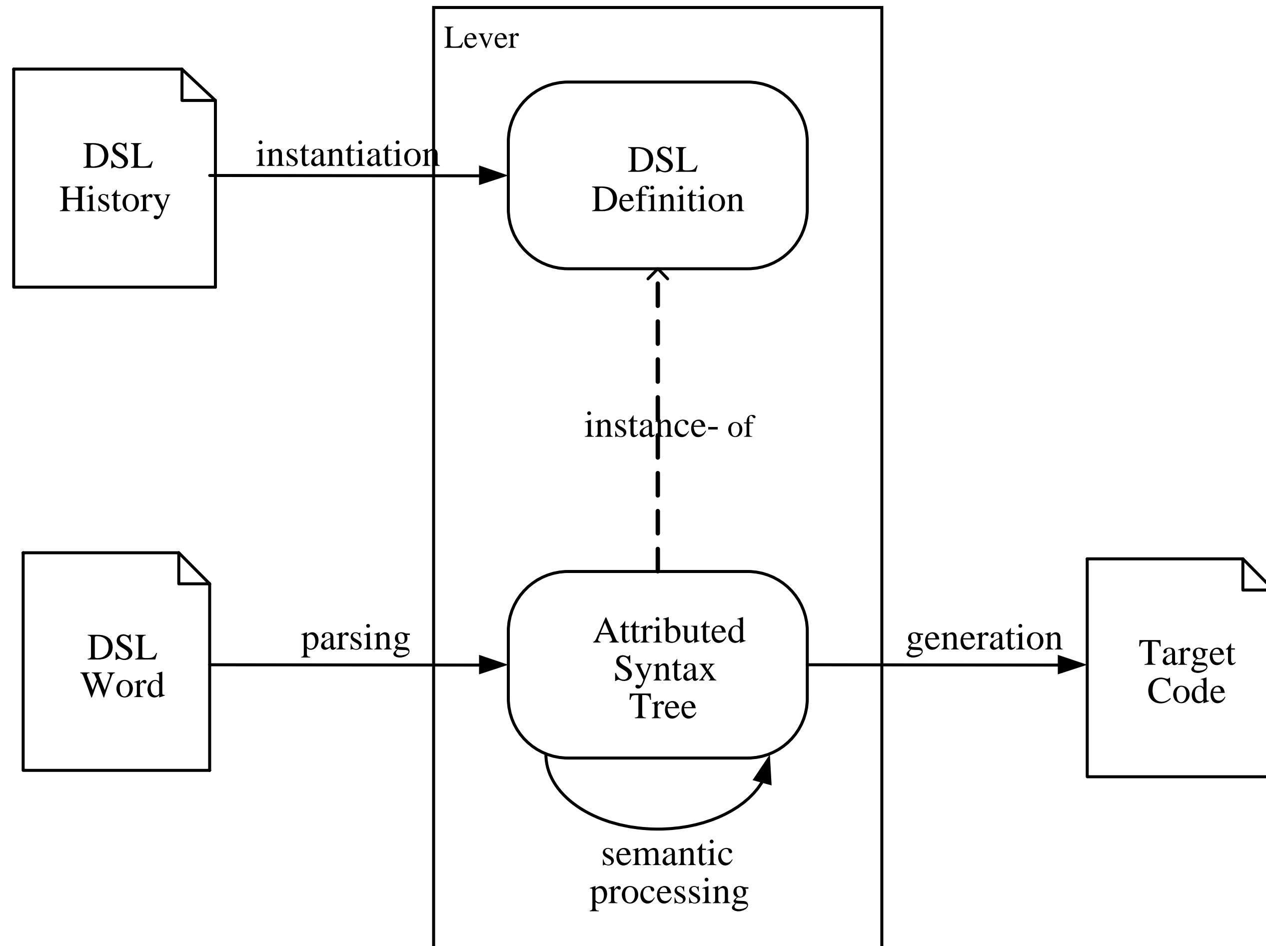
LEL = GEL ◦ WEL
- Coupled evolution operations

⇒ Higher level of abstraction
⇒ Reuse



WEL
- Transformation of syntax tree
- complete, set-oriented

Language Evolver



Implementation

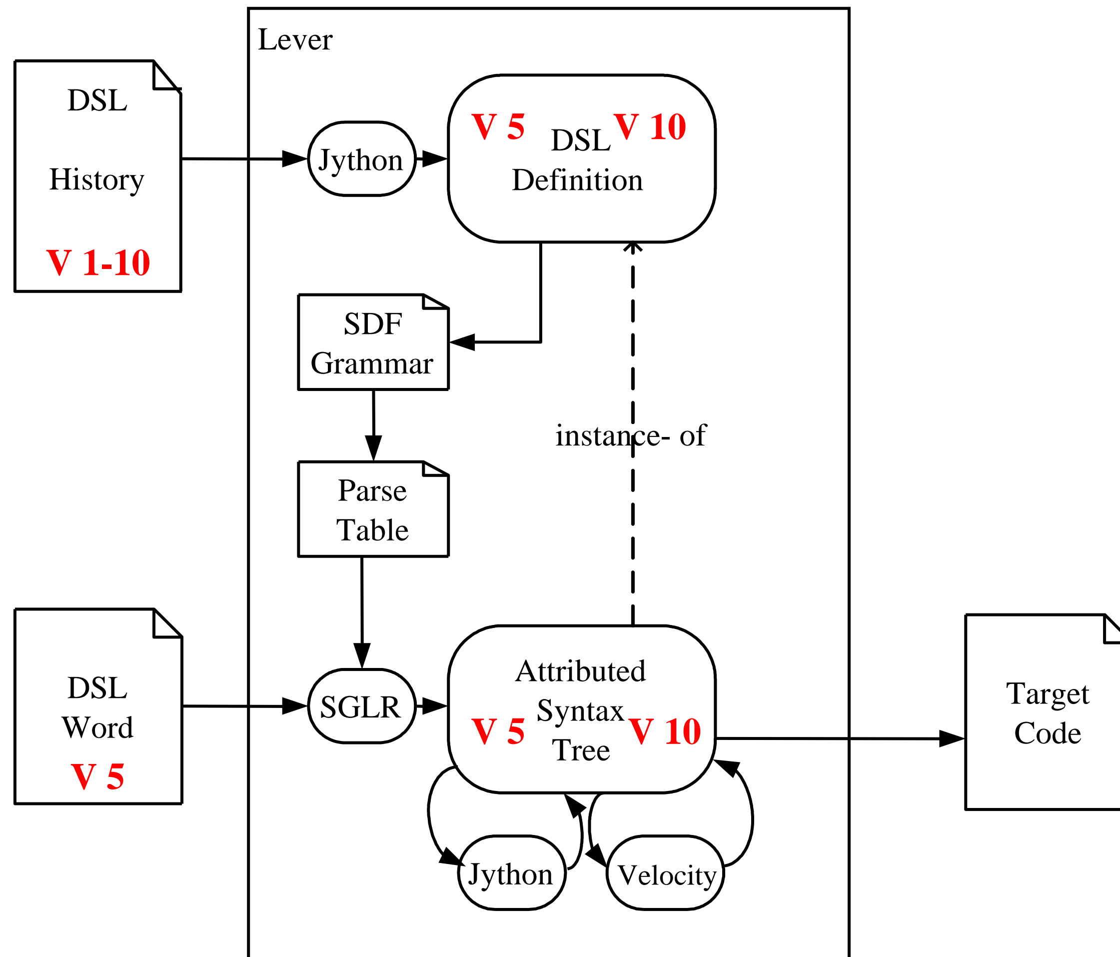
Definition Syntax and (translational) Semantics: **AXT**

- (Remote) OO- **A**ttribute Grammars
- **X**Path
- Code Generation **T**emplates

Tools / Technologies

- Parsing: *SDF* + *SGLR* (free of LL/LR like limitations)
- Code Generation: *Velocity Template Engine*
- XPath evaluation and checking: *Apache JXpath*
- Evolution languages: Internal DSLs in *Jython*

Translation Process



How could that look in practice?

Back to the case study.

CDL: Version 1

```
Drill {
    SinglelineText Headline;
      MultiLineText Description;
    MultiLineText Delivery;
}
```

CDL: Version 2

```
catalog {
  document Drill {
    field SinglelineText Headline;
    field MultiLineText Description;
    field MultiLineText Delivery;
  }
}
```

- Refactoring: Abstract syntax did not change
- Only coupled evolution operations

CDL: Version 3

```
catalog {
    document Drill {
        field SinglelineText Headline caption „Überschrift“;
        field MultiLineText Description caption „Beschreibung“;
        field MultiLineText Delivery caption „Lieferumfang“;
    }
}
```

- Language extension (local)
- Coupled evolution operation + manual extension of templates
- (Inserts default value for caption)

CDL: Version 4

```
catalog {
    document Drill {
        field SinglelineText Headline;
        field MultiLineText Description;
        field MultiLineText Delivery;
}}
language german {
  document Drill {
    field Headline caption „Überschrift“;
    field Description caption „Beschreibung“;
    field Delivery caption „Lieferumfang“;
}}
```

Summary

Idea: Bottom-Up DSL construction to accomodate variability

- (Coupled) DSL evolution operations, automate
  - Adaptation of Compilers
  - Migration of existing Programs

Approach

- DSL History: contains evolution operations for all consecutive language versions
- DSL History Interpretation -> Compile programs from all versions

Limitations

Grammars (systematic)

- Well understood, few, clean concepts => Seemed simple…..
- LL/LR grammars not closed under conjunction / extension
- GLR grammars: ambiguity undecidable
- GLR parsers: bad error handling / unfriendly error messages

Limitations of Lever (our fault)

- Academic Prototype (Implementation not very forgiving)
- Limited grammar engineering tool support

Outlook

Feasibility?

- How much can really be automated?

=> Study of evolution history of 3 large industrial MMs

Formalism?

- Which tools could be better suited?

=> MetaModels instead of grammars: EMF

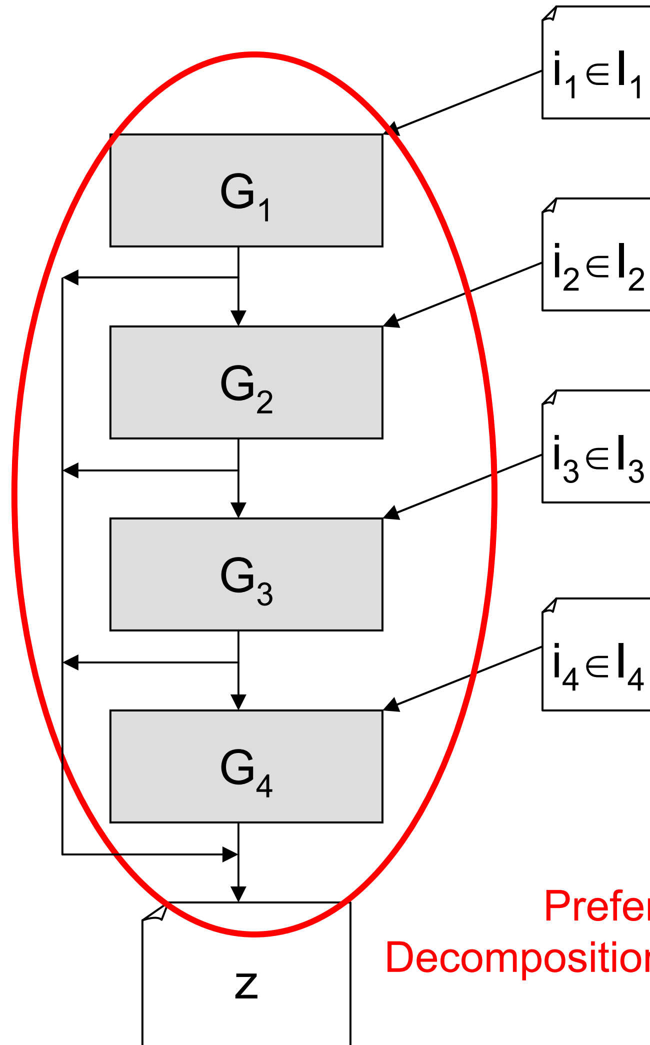
Does this really work in practice?

- Master thesis will implement CDL at company in germany

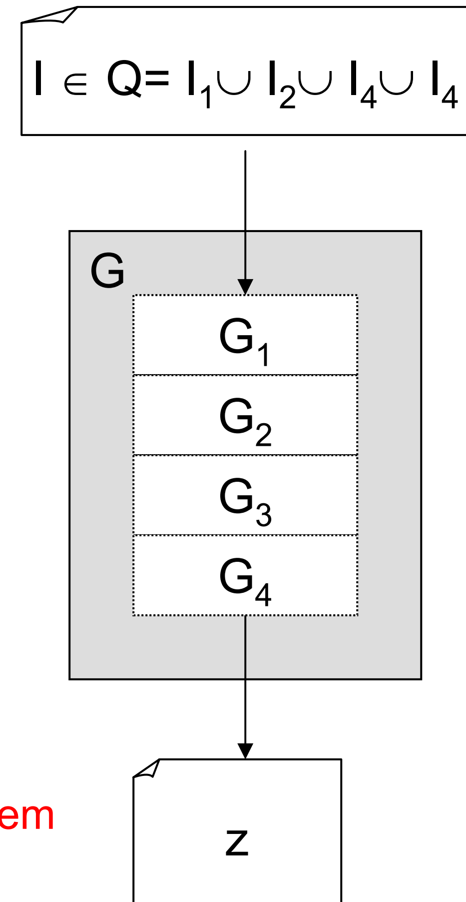
Questions?

Thank You!

Multi-Level Generation



=



Preferable approach
Decomposition of translation problem

Maintenance of Multi-Level Generator

- Output of higher-level language is input of lower level lang.

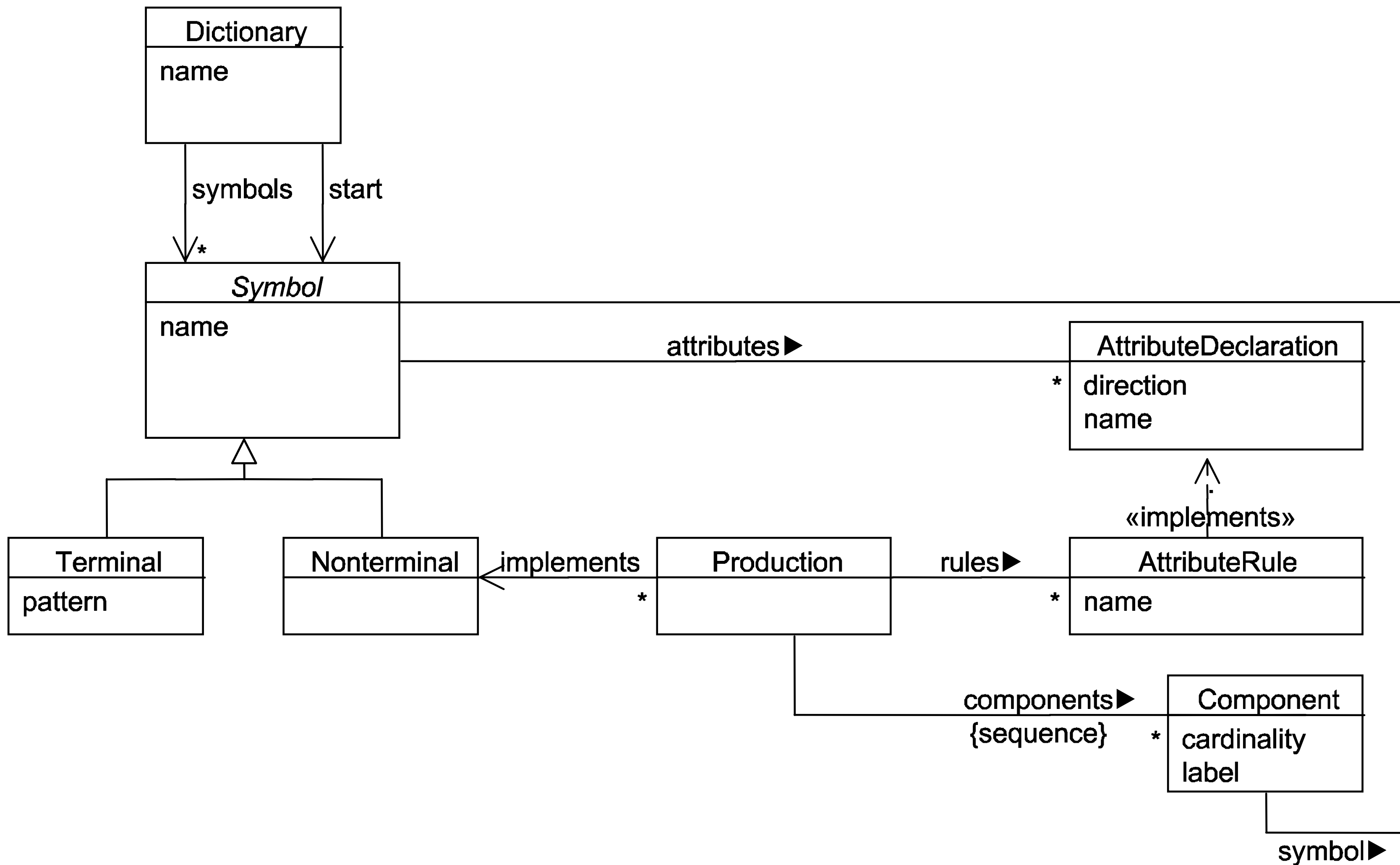
$\Rightarrow$ Evolution on higher level potentially affects all levels beneath it.

- Change of Input language in lower level must be reflected by output language of higher level.

$\Rightarrow$ Evolution on lower level potentially affects levels above it.

Next goal: Check output of higher level against input of lower level. („Does generated code always compile?“)

DSL Dictionary Metamodel



DSL Dictionary - Beispiel