TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

Towards Well-formed Fragment Composition with Reference Attribute Grammars

Sven Karol, Christoff Bürger and Uwe Aßmann

CBSE’12, Bertinoro, 26-06-2012

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

Background and Motivation

Basic Terminology[Kristensen+87, Aßmann 03]

Fragment Composition: methodology for *syntax-safe* source code composition according to the language grammar or metamodel.

- ❑ a Basic implementation technique for syntax-safe templates, code generation, aspect-oriented programming systems,….

Fragment: partial or under-specified piece of *source code* of a program or model (e.g., method, field declaration, class, expression…)

Slot: Explicitly declared variation point in a fragment.

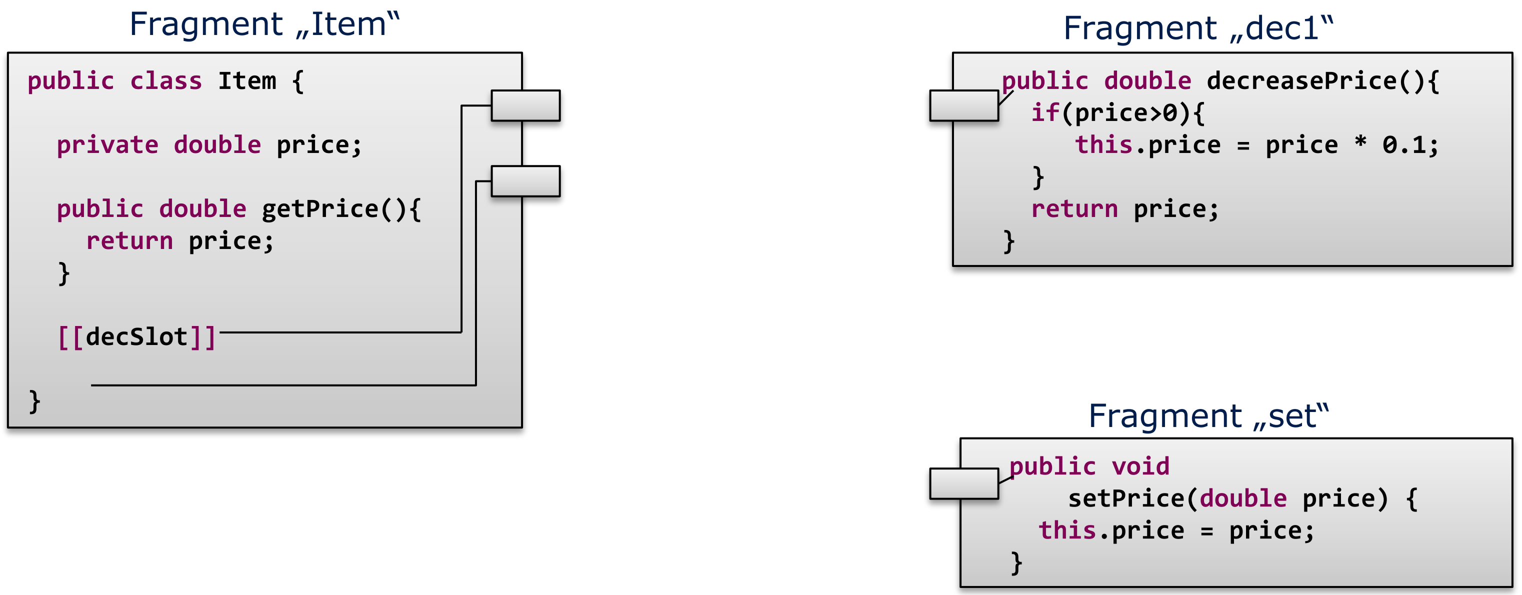
- ❑ can be bound to a syntactically compatible fragment

Hook: Implicit extension point in a fragment.

- ❑ can be extended with syntactically compatible fragments

Background and Motivation

Fragment Composition Example



Fragment „Item“

```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  [[decSlot]]

}
```

Fragment „dec1“

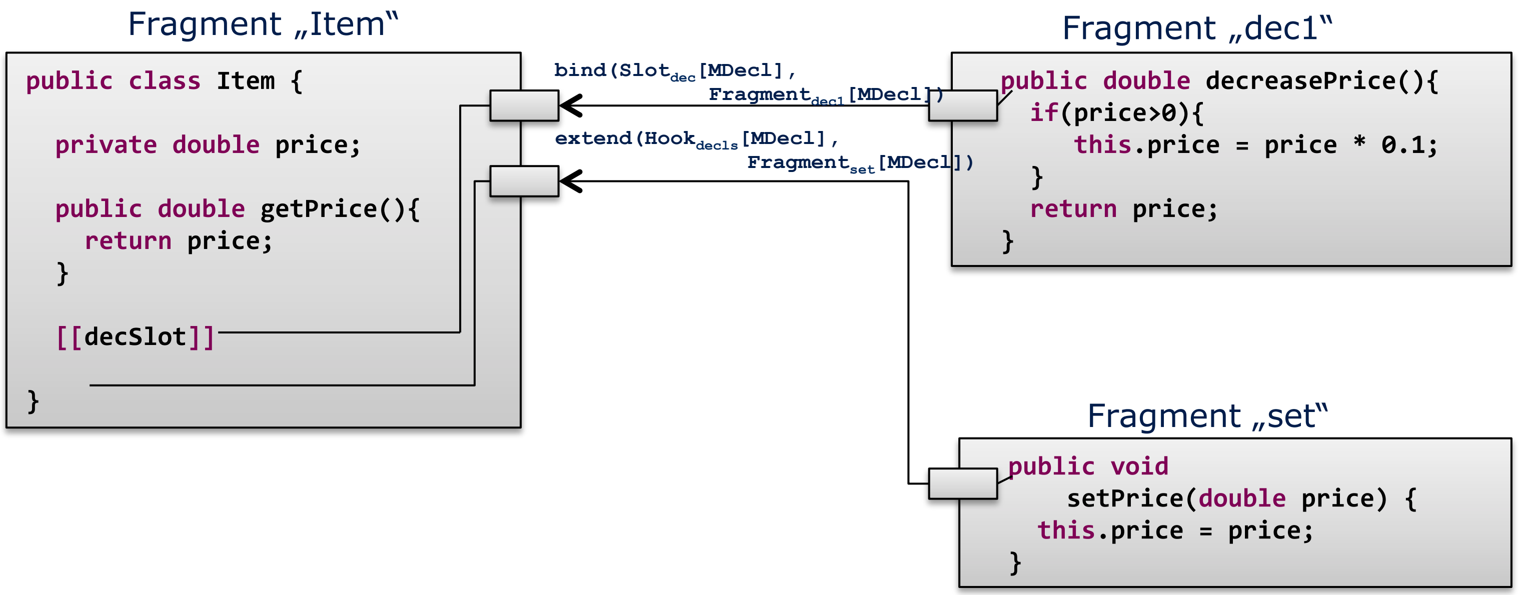
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

Fragment „set“

```
public void
     setPrice(double price) {
   this.price = price;
 }
```

Background and Motivation

Fragment Composition Example



Fragment „Item“

```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  [[decSlot]]

}
```

Fragment „dec1“

```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

Fragment „set“

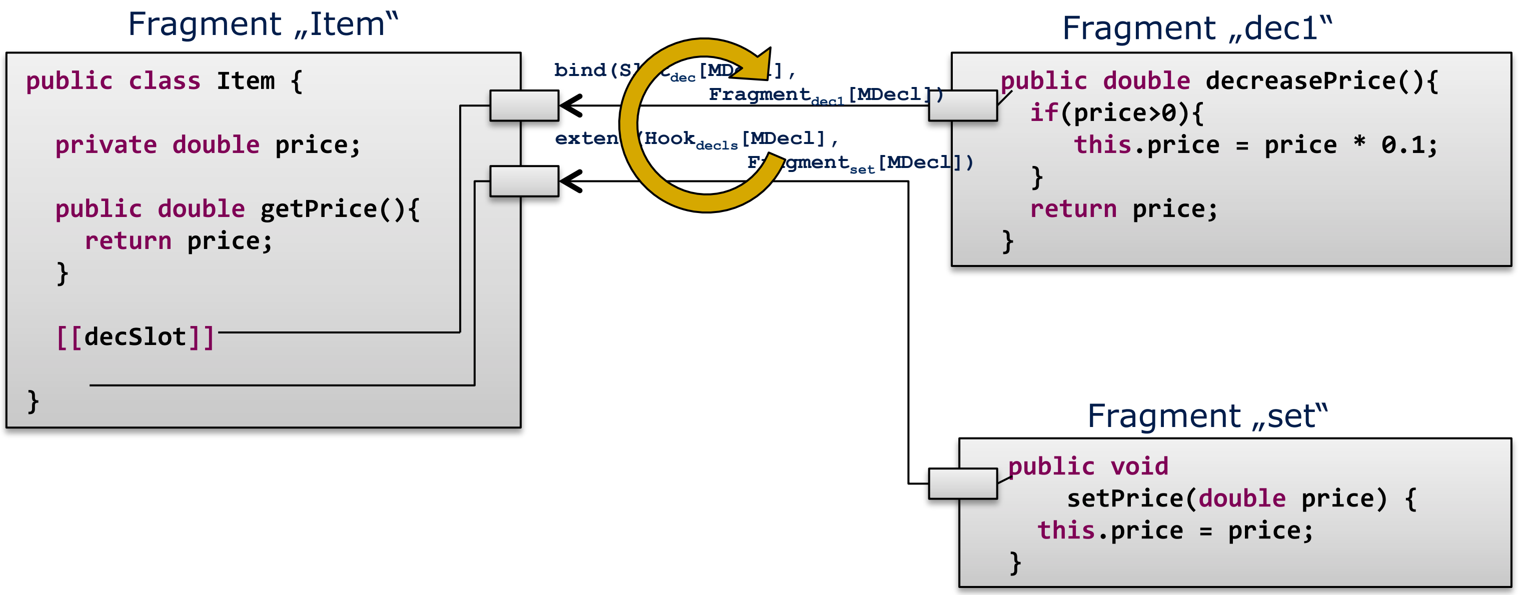
```
public void
    setPrice(double price) {
  this.price = price;
 }
```

bind(Slot$_{dec}$[MDecl], Fragment$_{dec1}$[MDecl])

extend(Hook$_{decls}$[MDecl], Fragment$_{set}$[MDecl])

Background and Motivation

Fragment Composition Example



bind(Slot$_{dec}$[MDecl], Fragment$_{decl}$[MDecl])
extend(Hook$_{decls}$[MDecl], Fragment$_{set}$[MDecl])

Fragment „Item“

```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  [[decSlot]]

}
```

Fragment „dec1“

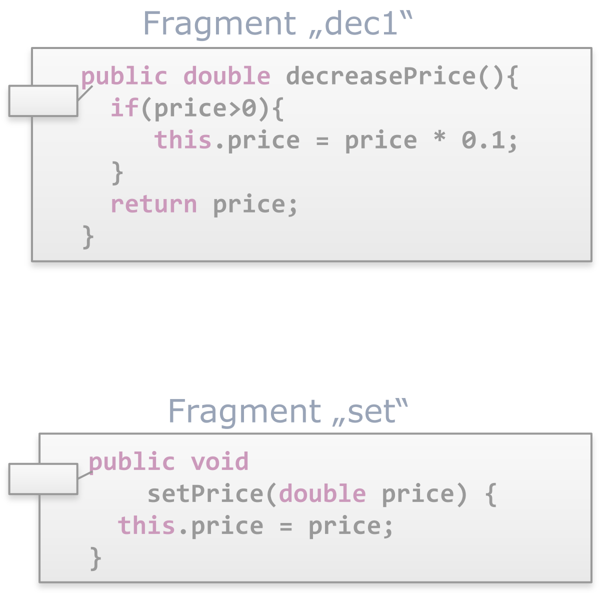
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

Fragment „set“

```
public void
    setPrice(double price) {
  this.price = price;
 }
```

Background and Motivation

Fragment Composition Example



Fragment „Item“

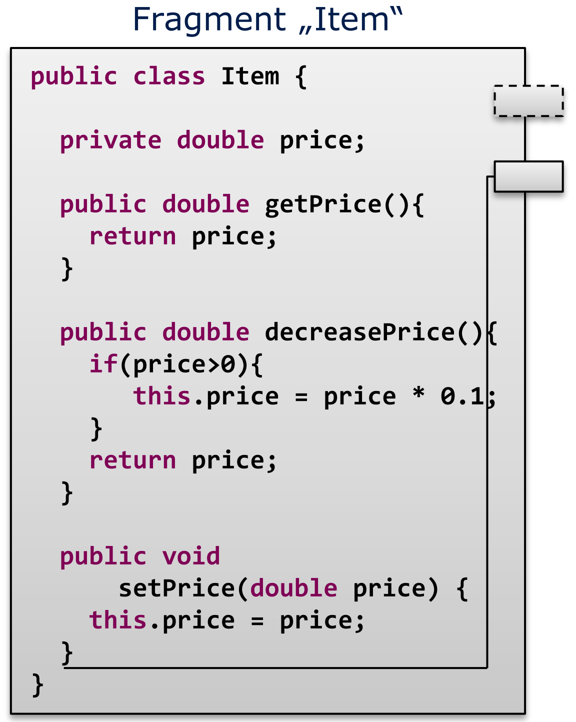
```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  public double decreasePrice(){
    if(price>0){
       this.price = price * 0.1;
    }
    return price;
  }

  public void
      setPrice(double price) {
    this.price = price;
  }
}
```

Fragment „dec1“

```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

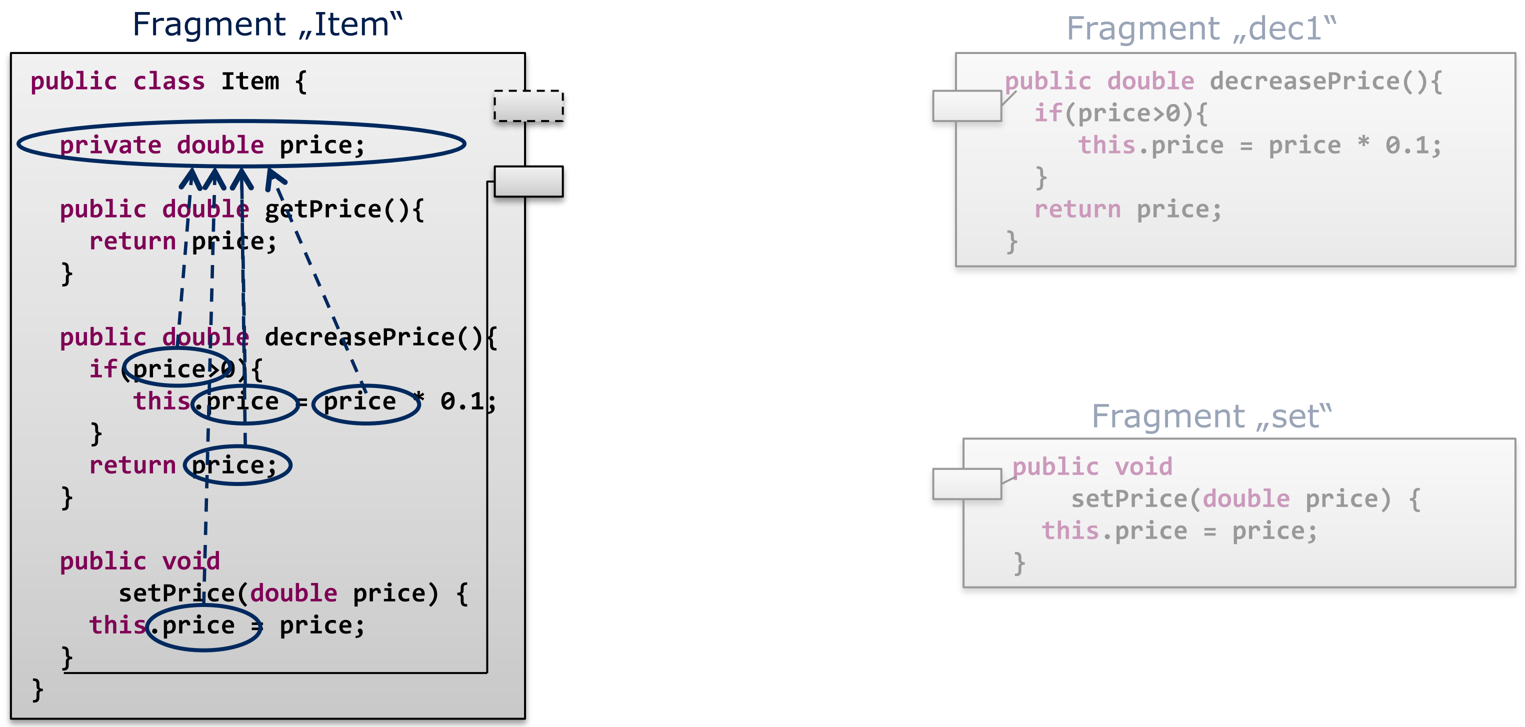
Fragment „set“

```
public void
    setPrice(double price) {
  this.price = price;
}
```



Background and Motivation

Fragment Composition Example


Fragment „Item“

```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  public double decreasePrice(){
    if(price>0){
       this.price = price * 0.1;
    }
    return price;
  }

  public void
      setPrice(double price) {
    this.price = price;
  }
}
```

Fragment „dec1“

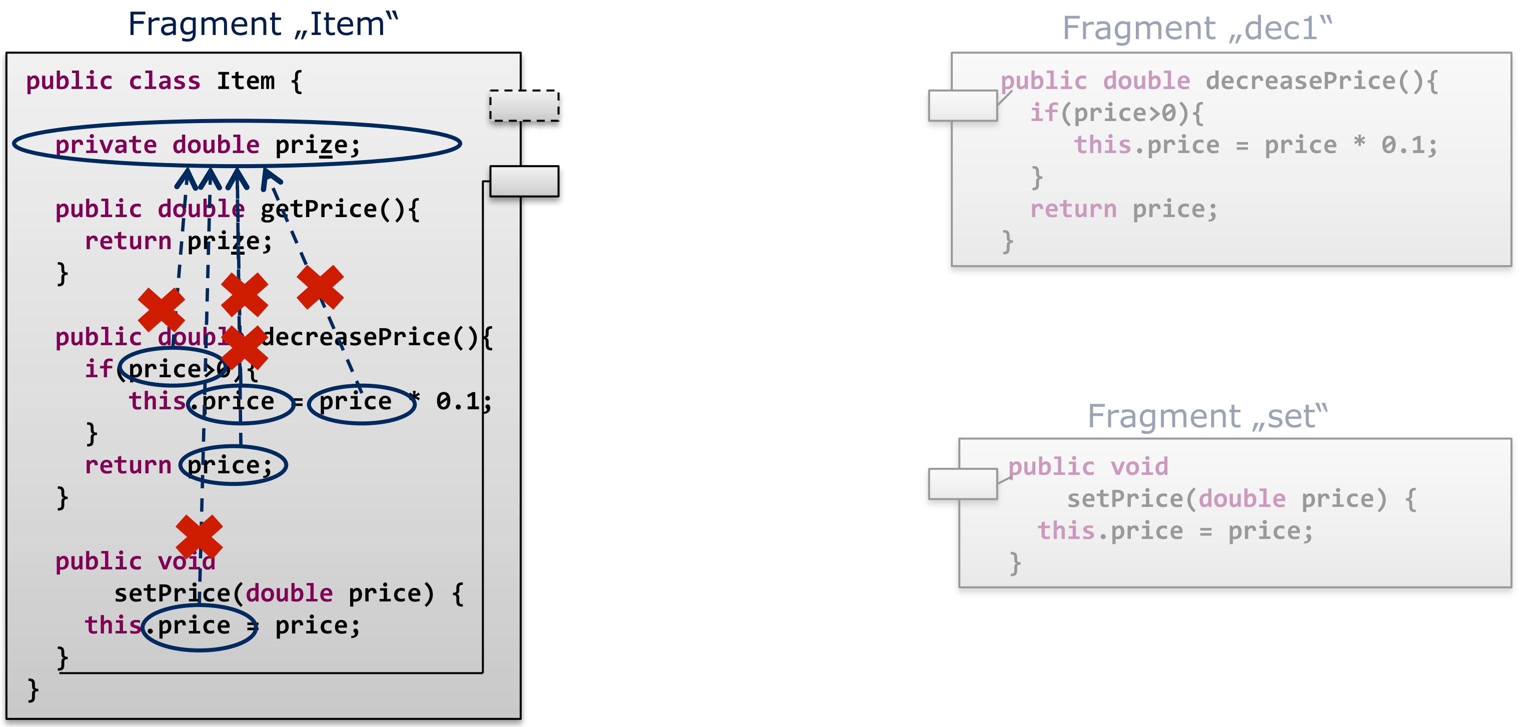
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

Fragment „set“

```
public void
    setPrice(double price) {
  this.price = price;
}
```

Background and Motivation

Fragment Composition Example



Fragment „Item"

```
public class Item {
  private double prize;

  public double getPrice(){
    return prize;
  }

  public double decreasePrice(){
    if(price>0){
       this.price = price * 0.1;
    }
    return price;
  }

  public void
      setPrice(double price) {
    this.price = price;
  }
}
```

Fragment „dec1"

```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

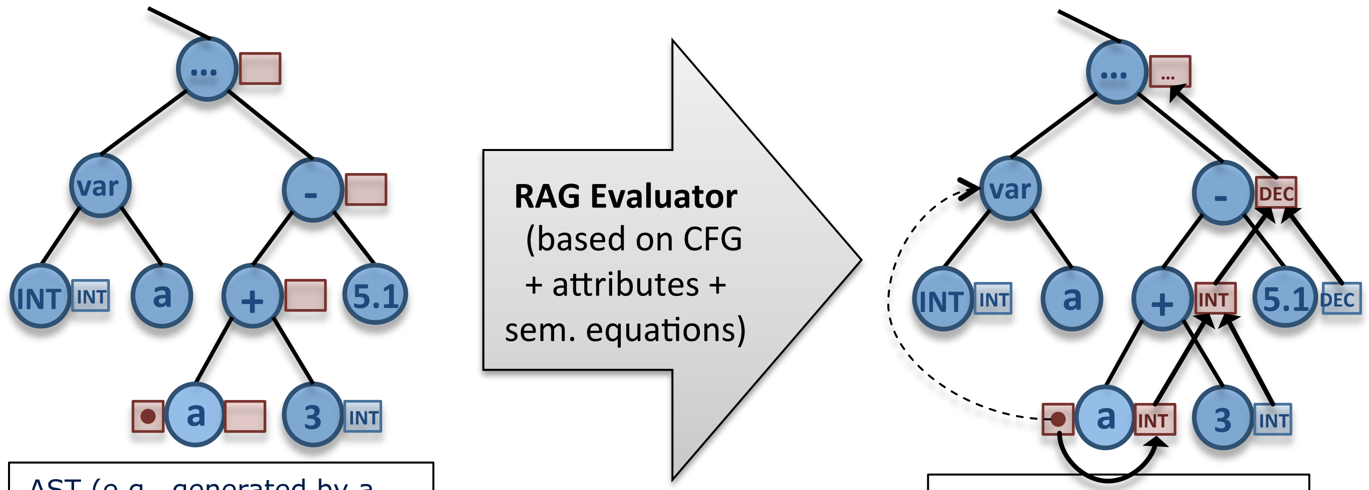
Fragment „set"

```
public void
    setPrice(double price) {
  this.price = price;
}
```

Background and Motivation

Solution Idea: Use Reference Attribute Grammars (RAGs) to specify fragment component models

- Formalism for specifying static semantics of programming languages and generating compiler frontends.
- Context-sensitive extension to context-free grammars/tree grammars:
  - ❑ non-terminals are assigned with (**inh**erited or **syn**thesized) *attributes*
  - ❑ for each context of an attribute (=grammar rule) a semantic equation specifies the attribute value

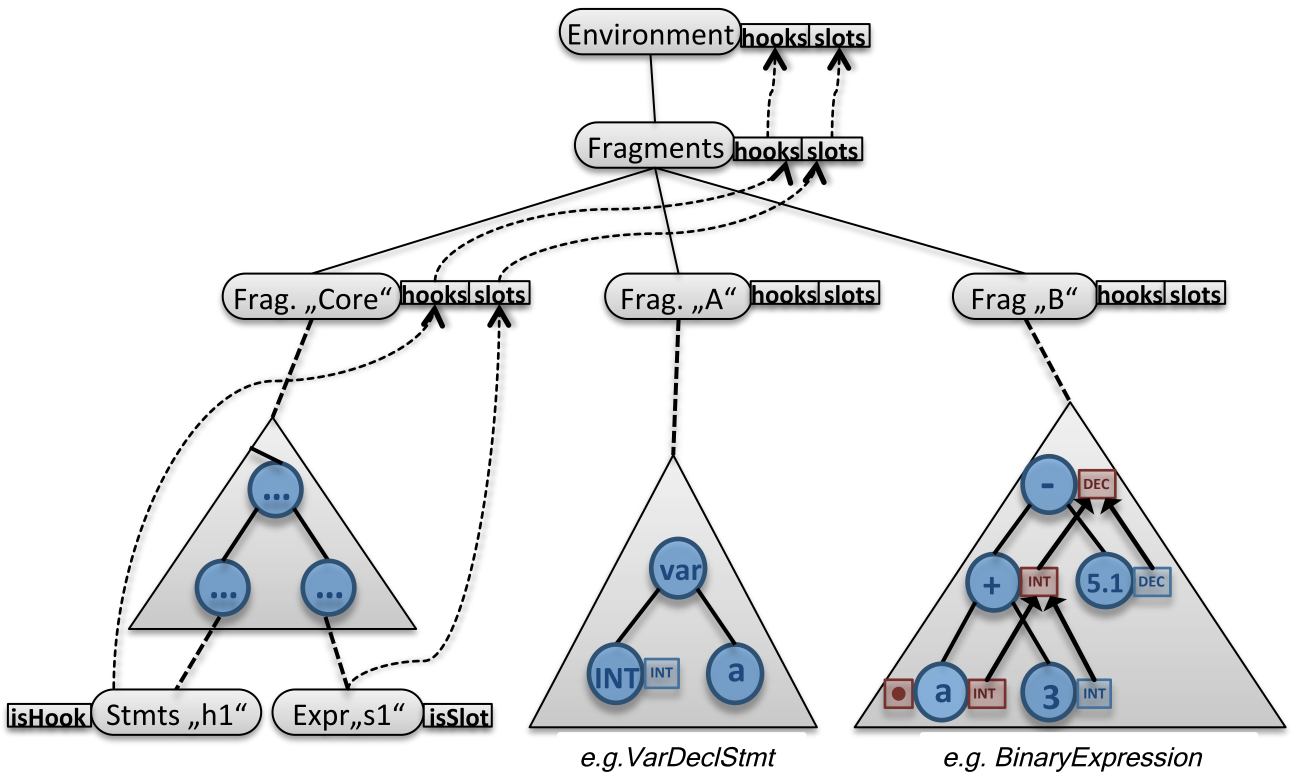


AST (e.g., generated by a parser from Java/C/.. code)

ASG (spanning tree with reference edges)

RAG-based Fragment Component Models

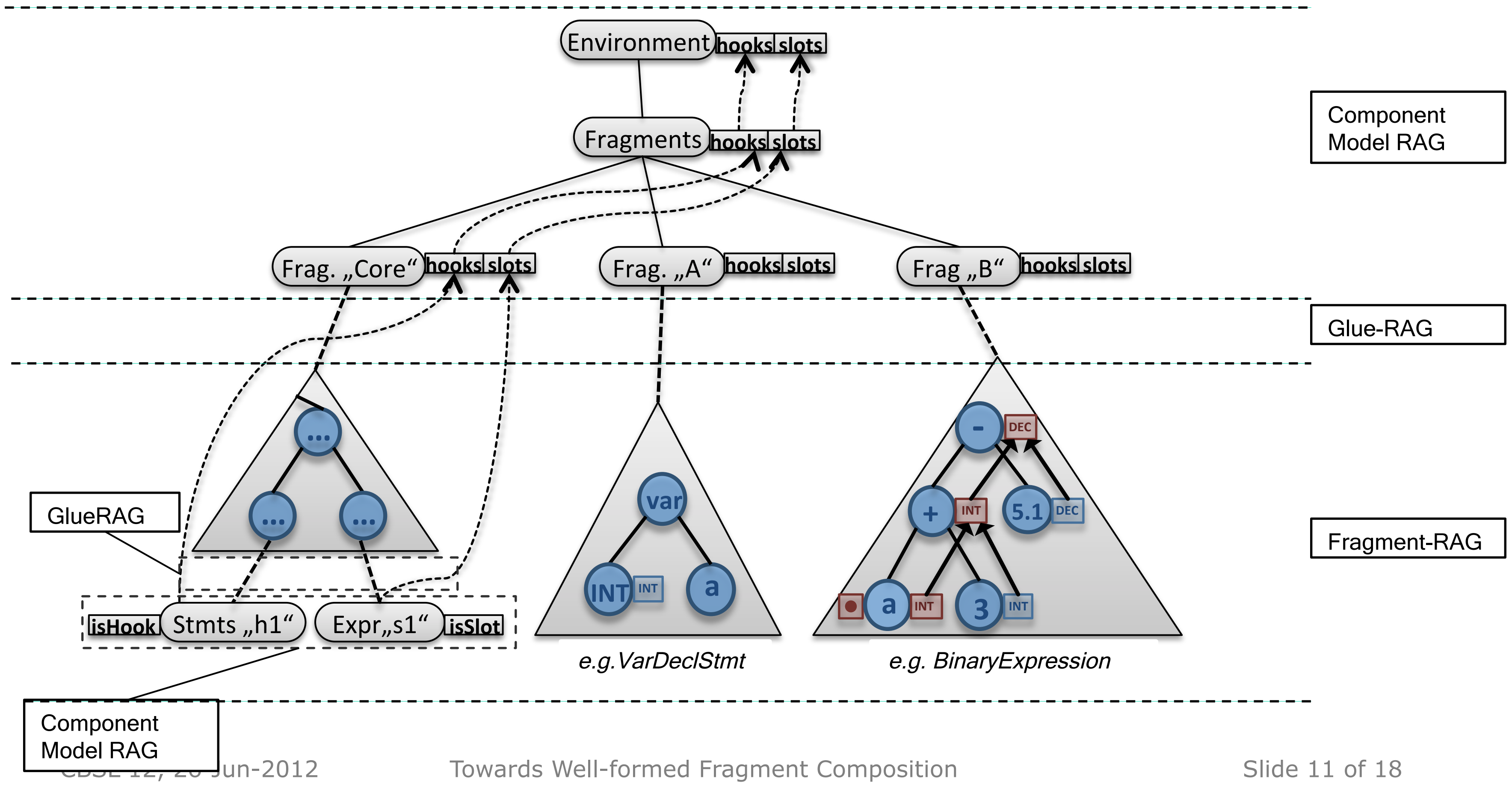
Example instance of a fragment component model



TECHNISCHE UNIVERSITÄT DRESDEN

RAG-based Fragment Component Models

Example instance of a fragment component model



TECHNISCHE UNIVERSITÄT DRESDEN

Fragment Contracts

Terminology

Fragment assertions are (automatically) derived static properties of a given (code) fragment.

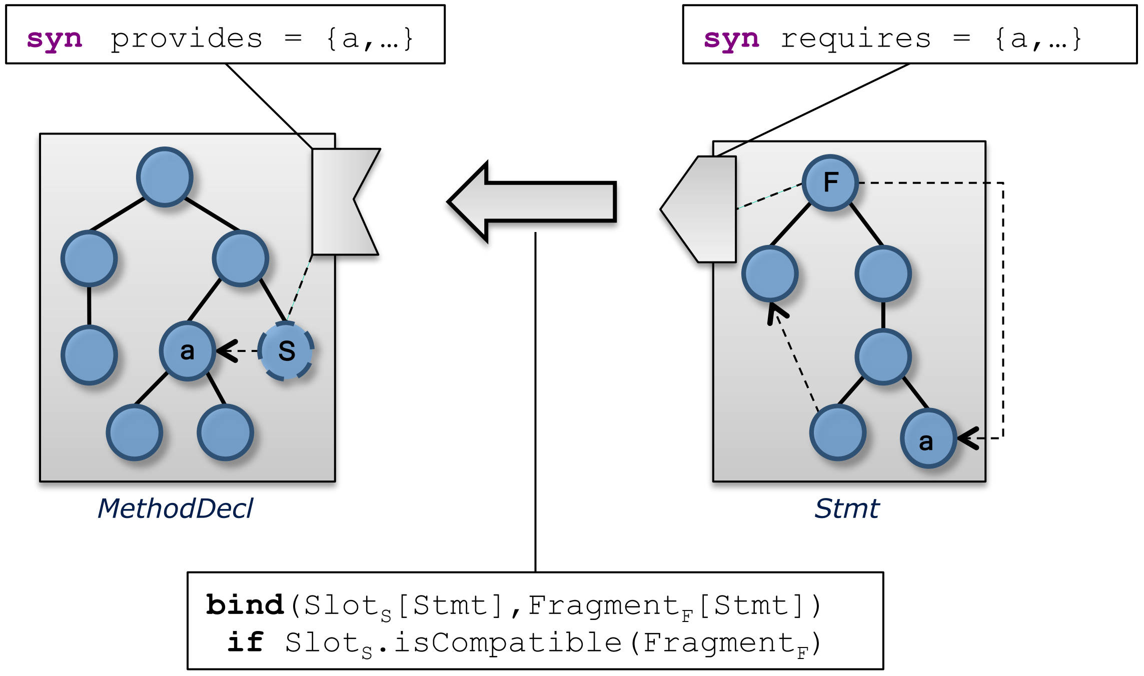
Fragment contracts are composition (pre-)conditions over fragment assertions.

- ❑ Ensure fragment compatibility w.r.t. static semantics and additional constraints
- ❑ Locate errors in composition programs
- ❑ Automatically select a compatible fragment component from a fragment repository

TECHNISCHE UNIVERSITÄT DRESDEN

Fragment Contracts

Example: Def-Use Relation



```
bind(Slot_S[Stmt],Fragment_F[Stmt])
 if Slot_S.isCompatible(Fragment_F)
```

Java Case Study

- Java-Fragments based on the RAG tool **JastAdd2** and the **JastAddJ** extensible Java compiler by Hedin/Ekman

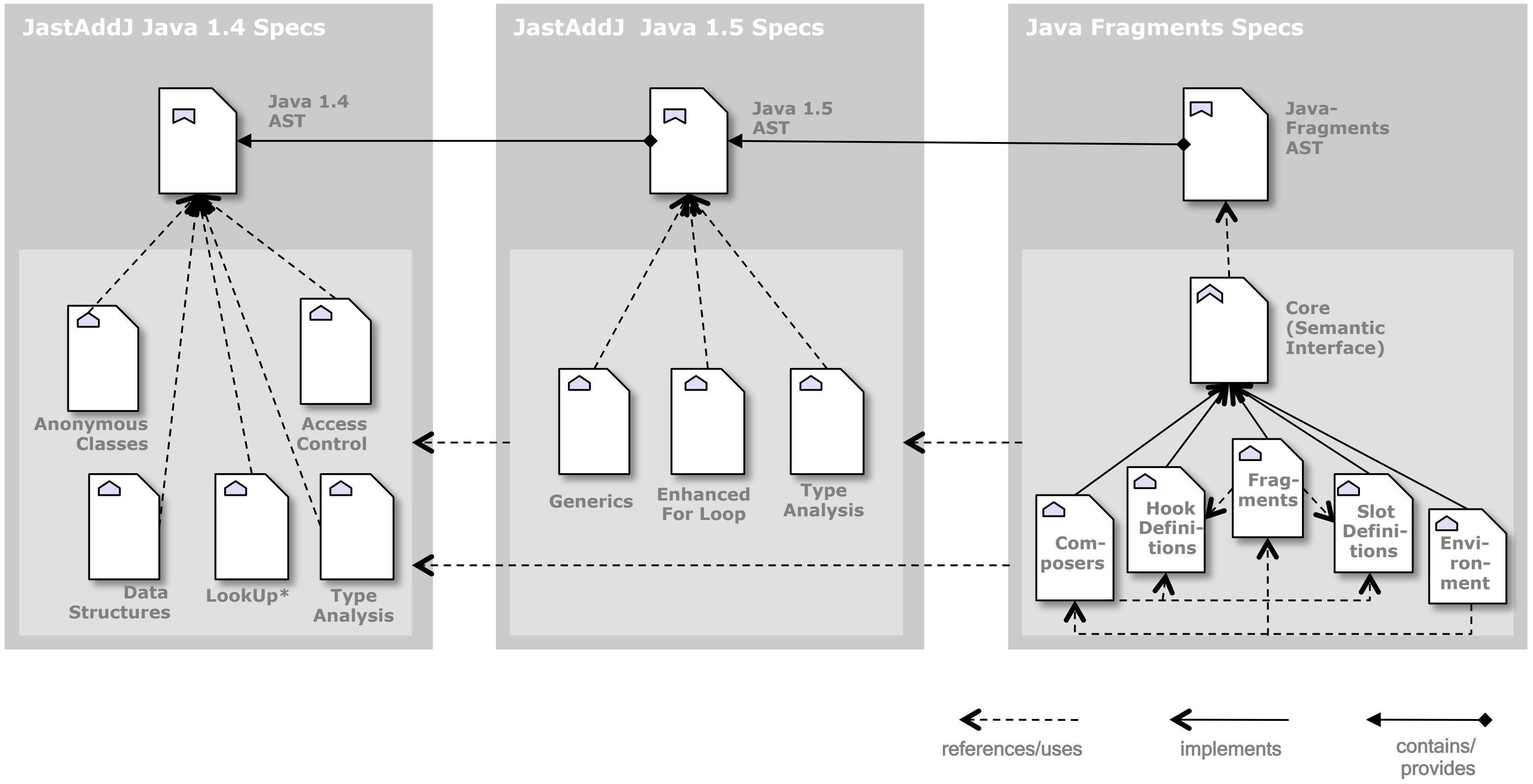
- JastAdd2 (www.jastadd.org)
  - ❑ Supports reference, higher-order and collection attributes, and rewrites
  - ❑ Supports OO ASTs and is implemented in Java
  - ❑ Supports extensible compiler construction approaches [Ekman06]
  - ❑ Generates Deman-driven evaluators with cached attributes

- JastAddJ
  - ❑ RAG based extensible Java compiler
  - ❑ Fully compliant with Java2 1.5
  - ❑ Modular Name + type analysis for Java
  - ❑ Bytecode reader + generator
  - ❑ Modular Java Grammar (basically LALR)
  - ❑ PrettyPrinter

TECHNISCHE UNIVERSITÄT DRESDEN

Java Case Study

Overview of the involved RAG specifications



Java Case Study

Fragment Composition Features

- Extended Java 1.5 Specification and parser
  - ❑ Slot Markup (types, expressions, statements, literals, methods, variable declarations)
  - ❑ Addressable Hooks (class-members, method hooks, block hooks in different classes, parameter lists)
  - ❑ According fragment types
  - ❑ RAG API for *fragment contracts*
  - ❑ Java API for creating composition programs (staged composition possible)
  - ❑ Implementation of composition operators with conditional AST rewrites (not shown in the paper)

Conclusion and Outlook

Benefits

- First approach for well-formed fragment composition
  - ❑ e.g., for generating safe template engines, AOP systems
- Universal approach that can be transferred to any language
  - ❑ like an “add-on”, if RAG frontend exists
- Founded in the RAG formalism

Open Issues/Outlook

- Complex usage/more industrial scenarios
  - ❑ we have a first prototype on architectural skeletons
- Transfer to model-based languages/ web languages
- Safe-C implementation
- (Non)confluency of composition steps
- Connection with composition languages / ADLs

[Hedin00] Hedin, Görel. 2000. Reference Attributed Grammars. *Informatica (Slovenia)* 24, Nr. 3: 301–317.

[Boyland05] Boyland, John T. 2005. Remote attribute grammars. *Journal of the ACM* 52, Nr. 4 (July): 627–687.

[Johannes11] Johannes, Jendrik. 2011. Component-Based Model-Driven Software Development. Dissertation/Ph.D. thesis, Technische Universität Dresden, Januar.

[Henriksson09] Henriksson, Jakob. 2009. A Lightweight Framework for Universal Fragment Composition—with an application in the Semantic Web. Dissertation/Ph.D. thesis, Technische Universität Dresden, Januar.

[Aßmann03] Aßmann, Uwe. 2003. *Invasive Software Composition*. 1. Aufl. Springer.

[Bürger+11] Bürger, Christoff, Sven Karol, Christian Wende, und Uwe Aßmann. 2011. Reference Attribute Grammars for Metamodel Semantics. In *Software Language Engineering*. Springer Berlin / Heidelberg.

[Knuth68] Knuth, D. E. 1968. Semantics of context-free languages. *Theory of Computing Systems* 2, Nr. 2: 127–145.

[Vogt+89] Vogt, Harald H, Doaitse Swierstra, und Matthijs F Kuiper. 1989. Higher Order Attribute Grammars. In *PLDI '89*, 131–145. ACM.

[Ekman06] Ekman, Torbjörn. 2006. Extensible Compiler Construction. University of Lund.

[Grosch90] Grosch, Jan. 1990. *Object-Oriented Attribute Grammars*. Technical Report. Aachen: CoCoLab Datenverarbeitung, August 27.

[Hedin89] Hedin, Görel. An Object-Oriented Notation for Attribute Grammars. In *Proceedings of the 3rd European Conference on Object-Oriented Programming (ECOOP'89)*, 329-345. BCS Workshop Series. Nottingham, U.K.: Cambridge University Press.

[Meyer92] Meyer, B. 1992. Applying `design by contract'. *Computer* 25, Nr. 10 (Oktober): 40-51.

[Klaeren+01] Klaeren, Herbert, Elke Pulvermüller, Awais Rashid, und Andreas Speck. 2001. Aspect Composition Applying the Design by Contract Principle. In *Generative and Component-Based Software Engineering,* 2177:57-69. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

[Arnoldus11] Arnoldus, B. J. 2011. An Illumination of the Template Enigma: Software Code Generation with Templates. Technische Universiteit Eindhoven.

[Heidenreich+09] Heidenreich, Florian, Jendrik Johannes, Mirko Seifert, Christian Wende, und Marcel Böhme. 2009. Generating Safe Template Languages. In *Proc. of ACM 8th International Conference on Generative Programming and Component Engineering (GPCE'09)*. ACM Press.

[Bürger+10] Bürger, Christoff, Sven Karol, und Christian Wende. 2010. Applying attribute grammars for metamodel semantics. In *Proceedings of the International Workshop on Formalization of Modeling Languages*, 1:1–1:5. FML '10. New York, NY, USA: ACM.

[Ekman+07] The jastadd extensible java compiler. In *SIGPLAN Not.*, 42(10), S.1—18.

[Kristensen+87] Kristensen, Bent Bruun, Ole Lehrmann Madsen, Birger Moller-Pedersen, und Kristen Nygaard. 1987. „The BETA programming language". In *Research directions in object-oriented programming*, 7–48. Cambridge, MA, USA: MIT Press.

»Wissen schafft Brücken.«