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

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.

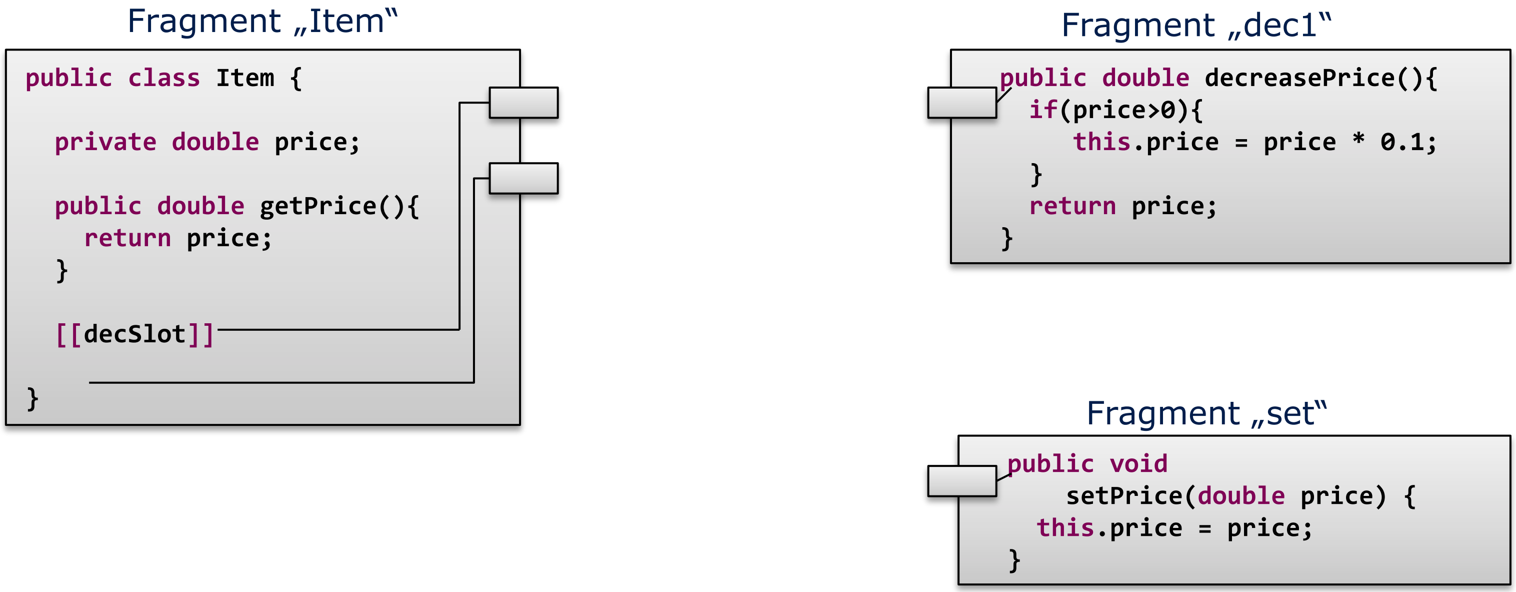
- ❑ can be bound to a syntactically compatible fragment

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"

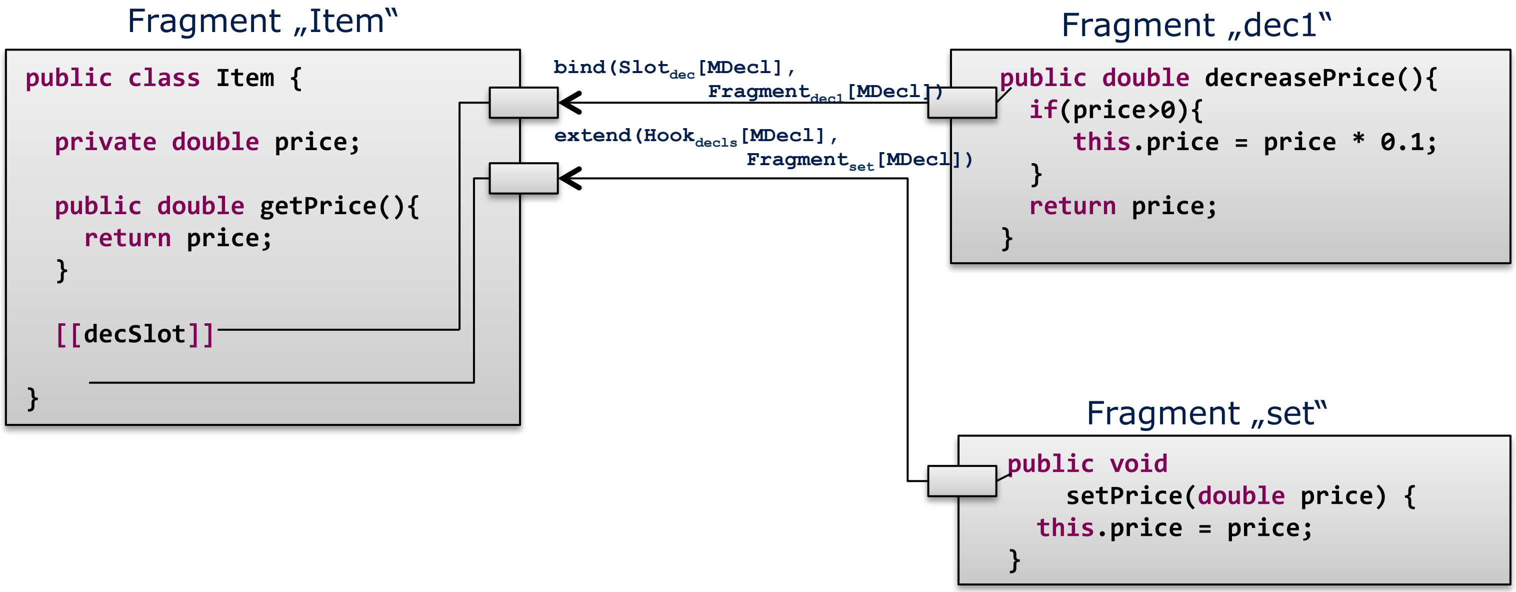
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

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]]

}
```

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

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

Fragment „dec1"

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

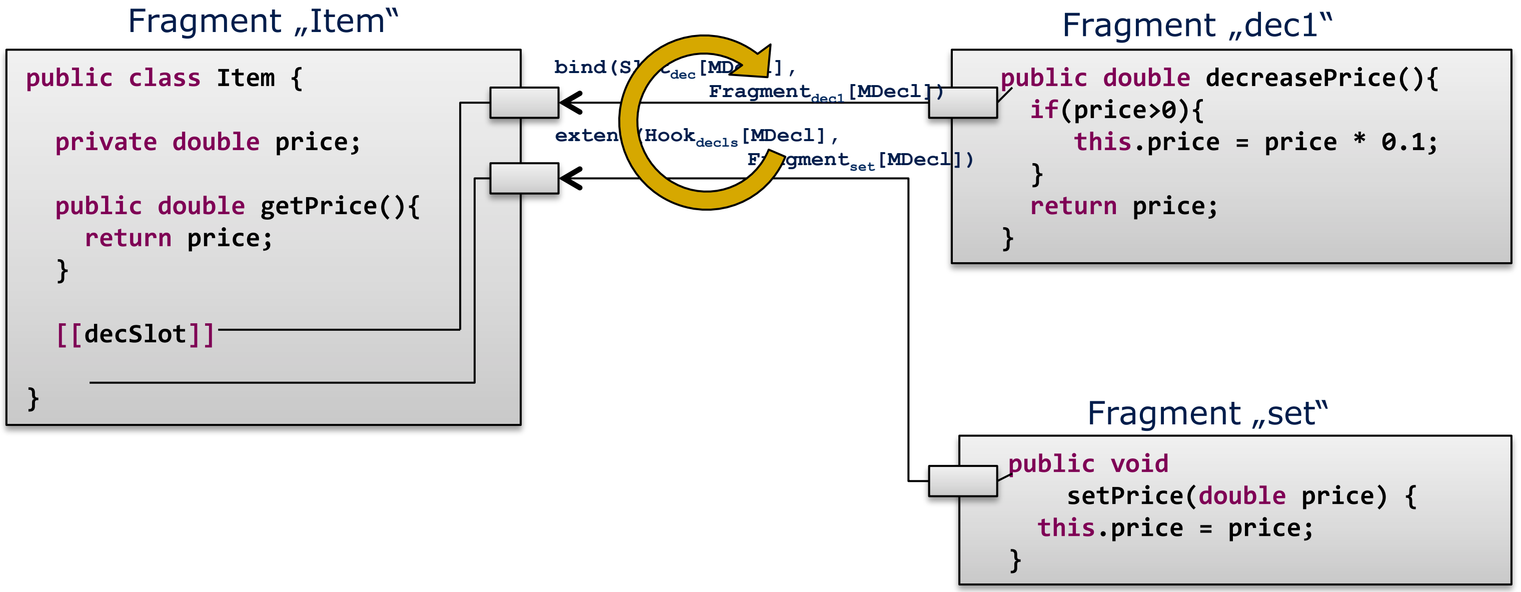
Fragment „set"

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

TECHNISCHE
UNIVERSITÄT
DRESDEN

Background and Motivation

Fragment Composition Example



Fragment „Item“

```
public class Item {

  private double price;

  public double getPrice(){
    return price;
  }

  [[decSlot]]

}
```

$\mathtt{bind(Slot_{dec}[MDecl],\ Fragment_{decl}[MDecl])}$

$\mathtt{extend(Hook_{decls}[MDecl],\ Fragment_{set}[MDecl])}$

Fragment „dec1“

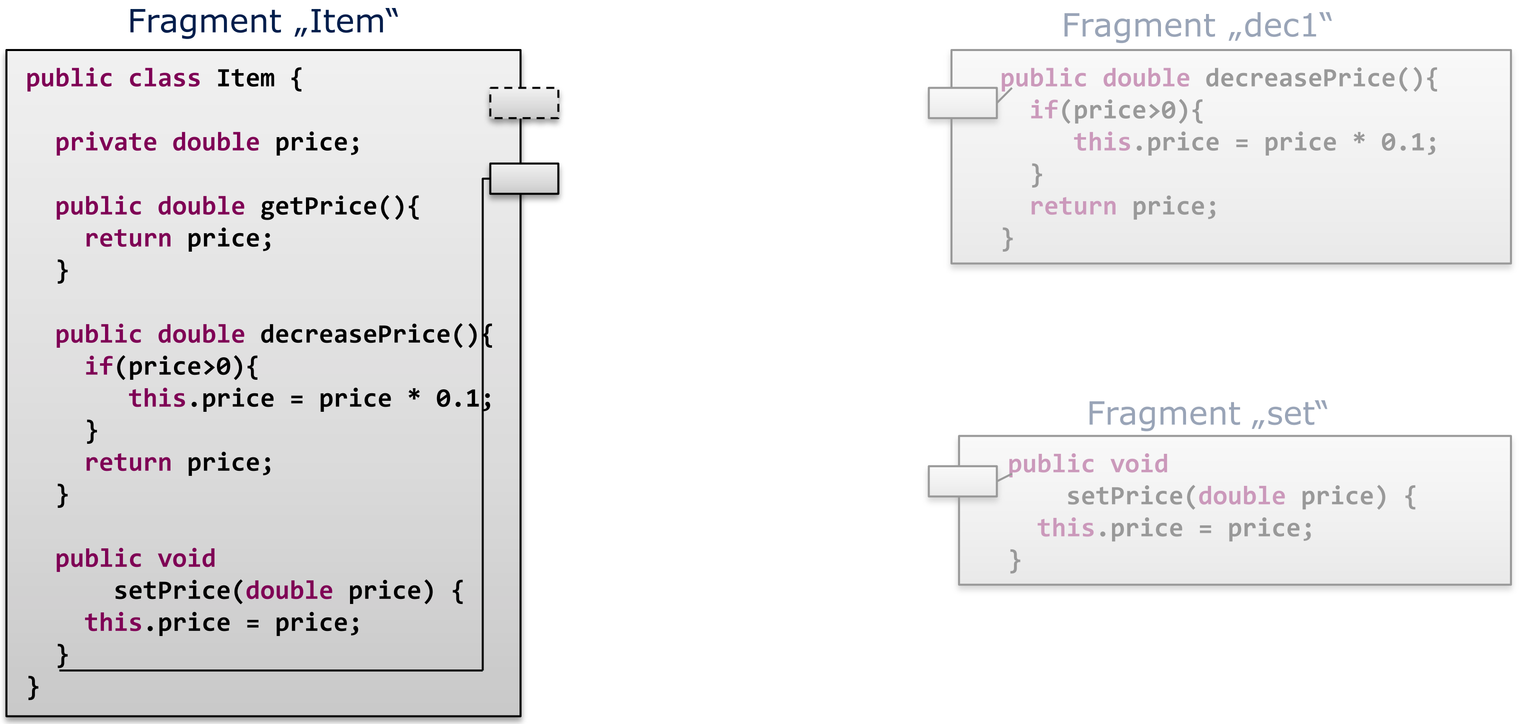
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

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“

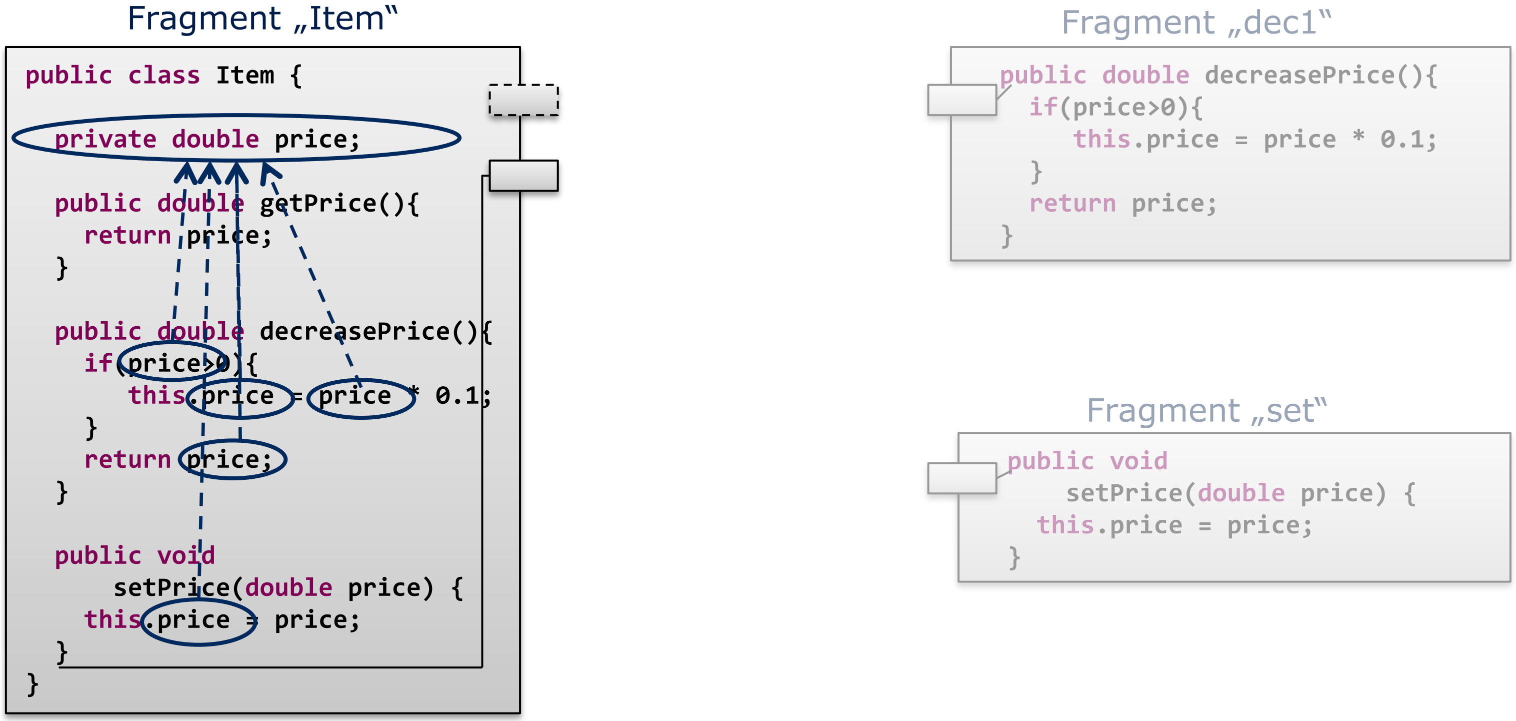
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

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“

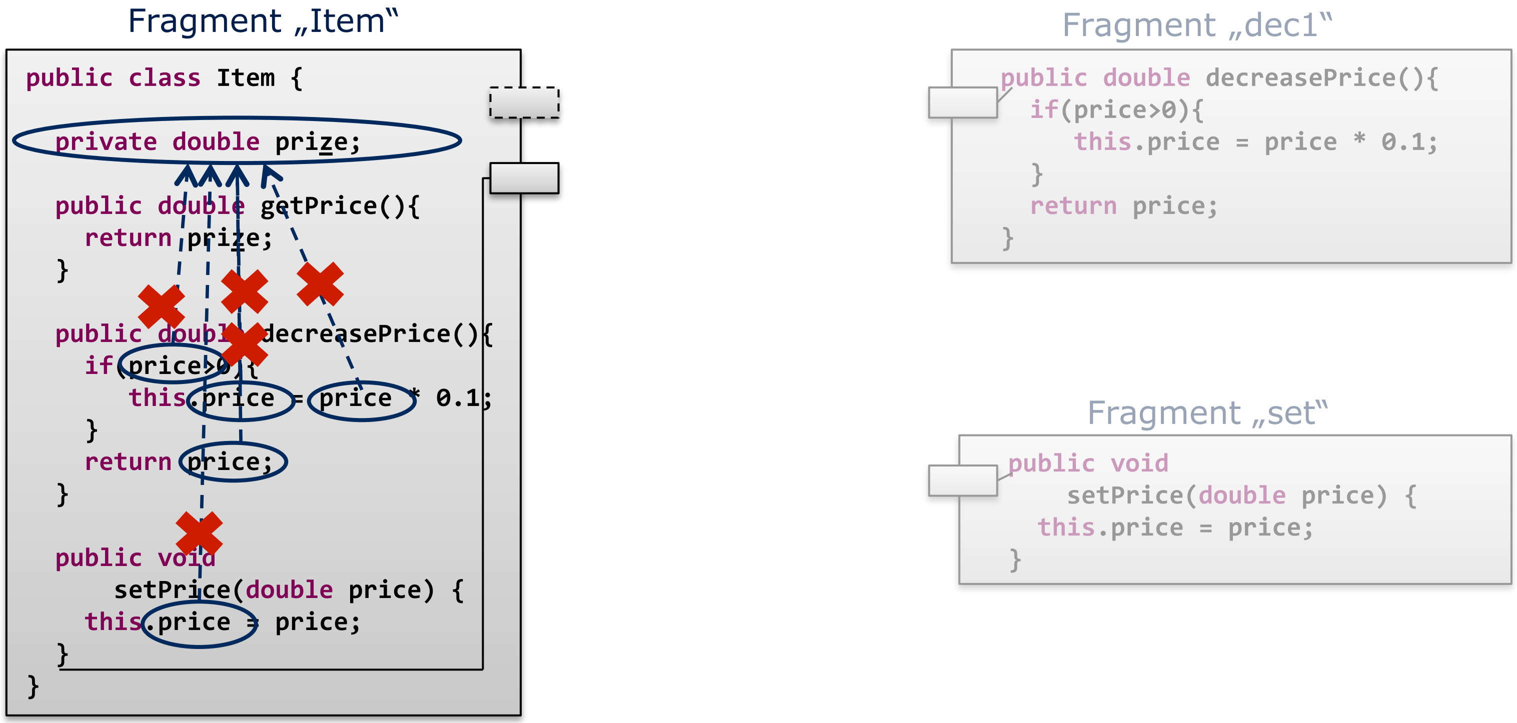
```
public double decreasePrice(){
  if(price>0){
     this.price = price * 0.1;
  }
  return price;
}
```

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;
}
```

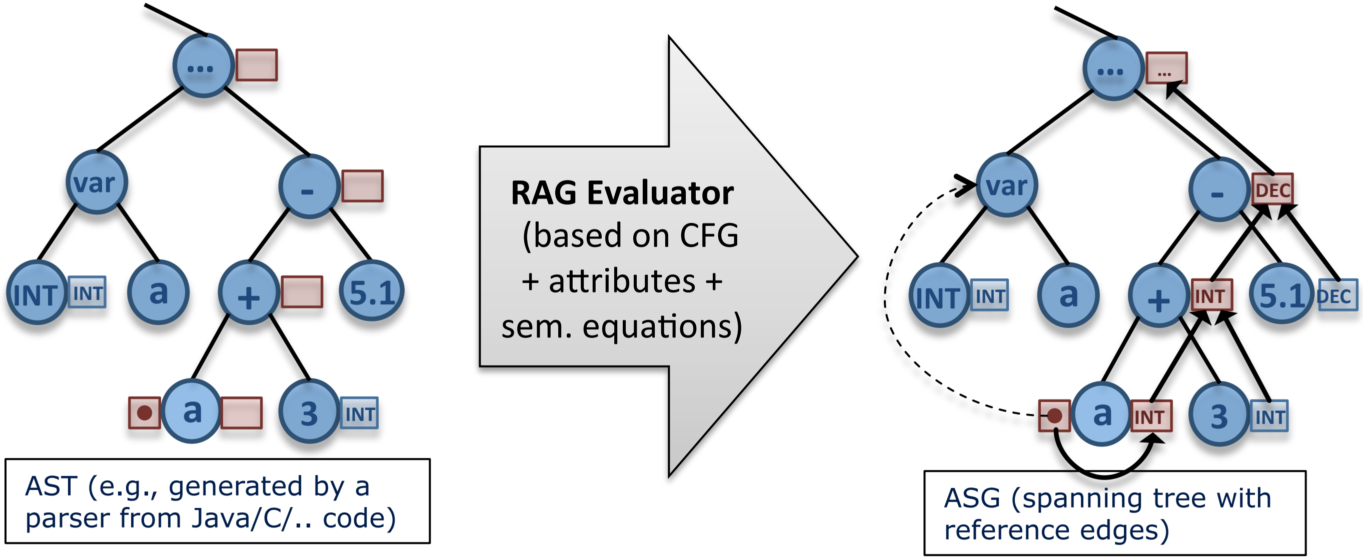
Fragment „set“

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

Background and Motivation

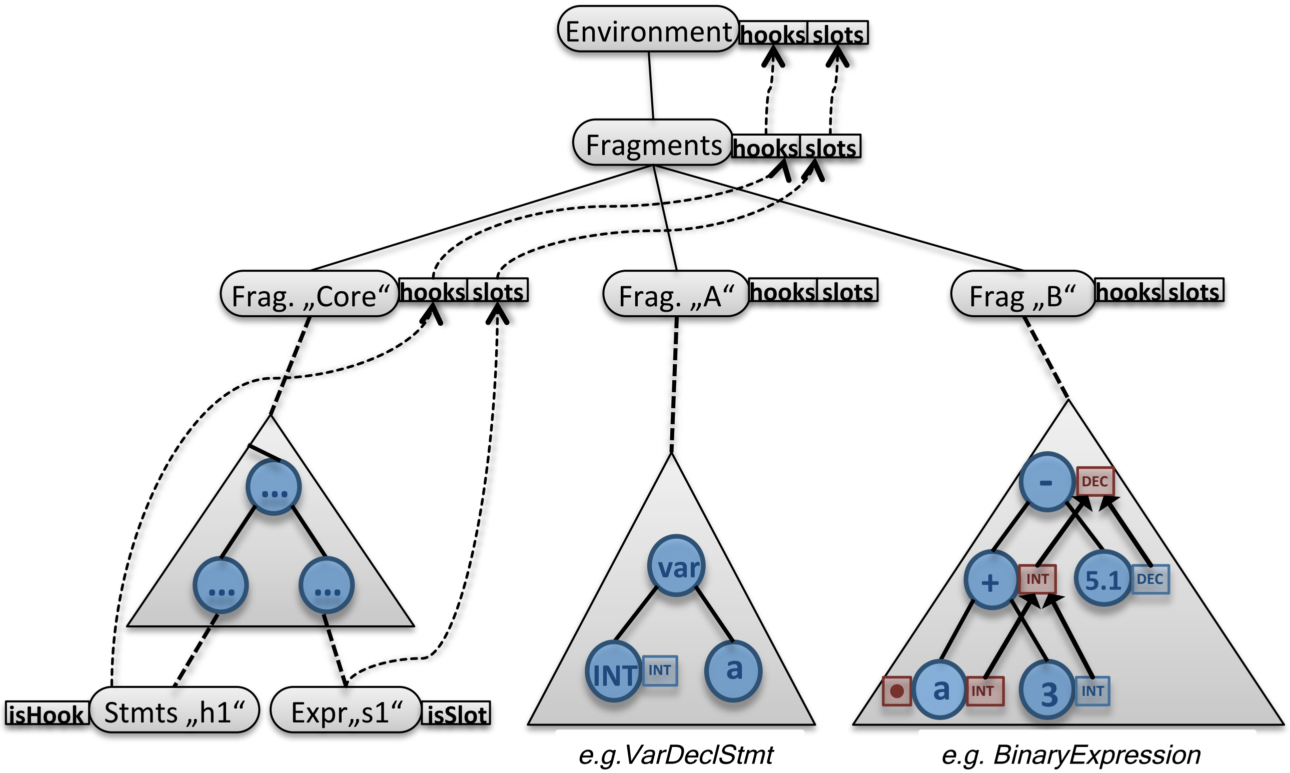
Solution Idea: Use Reference Attribute Grammars (RAGs) to specify fragment component models

- 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


...
var
-
INT
INT
a
+
5.1
a
3
INT
AST (e.g., generated by a parser from Java/C/.. code)
RAG Evaluator (based on CFG + attributes + sem. equations)
...
...
var
-
DEC
INT
INT
a
+
INT
5.1
DEC
a
INT
3
INT
ASG (spanning tree with reference edges)

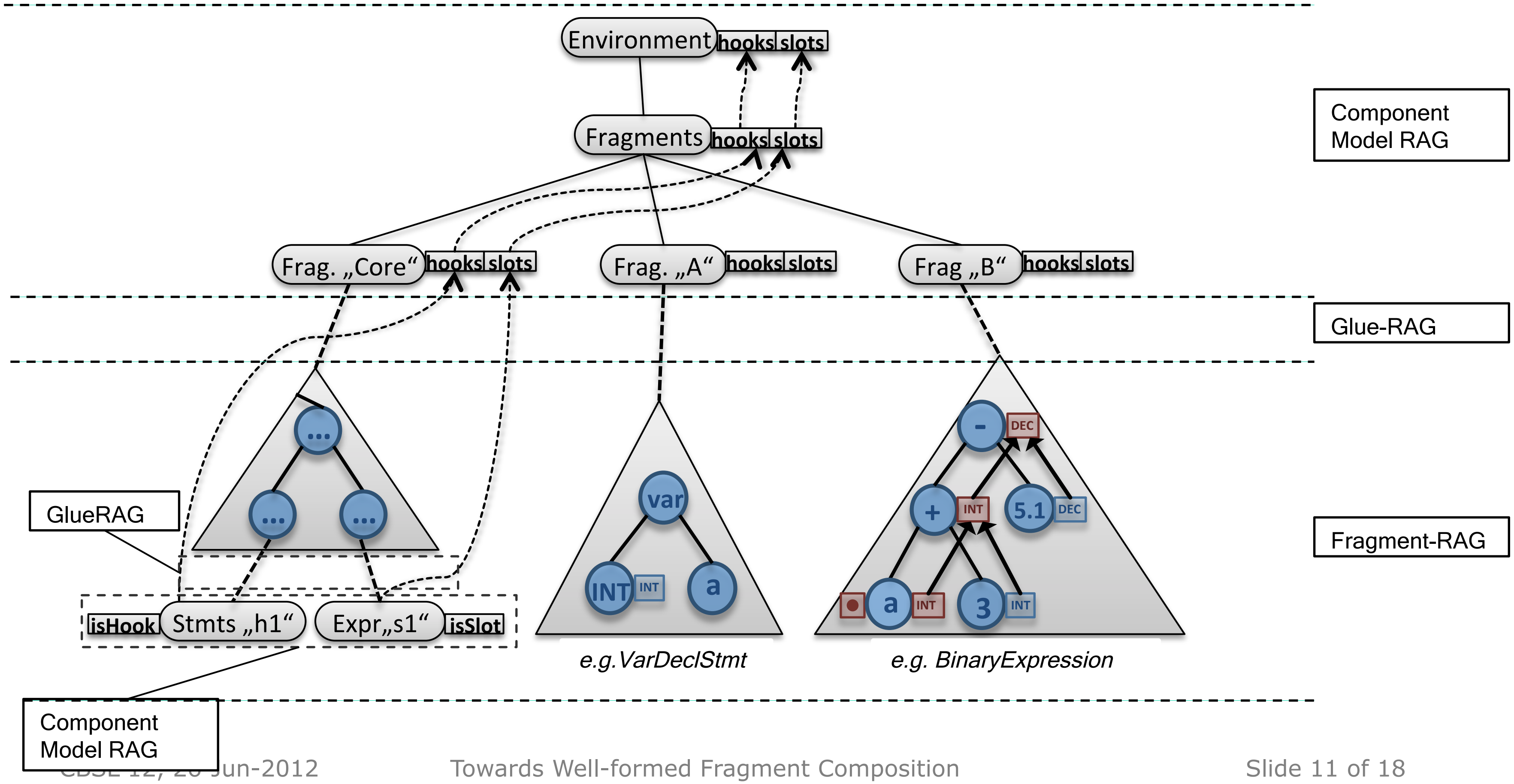
RAG-based Fragment Component Models

Example instance of a fragment component model



RAG-based Fragment Component Models

Example instance of a fragment component model



Fragment Contracts

Terminology

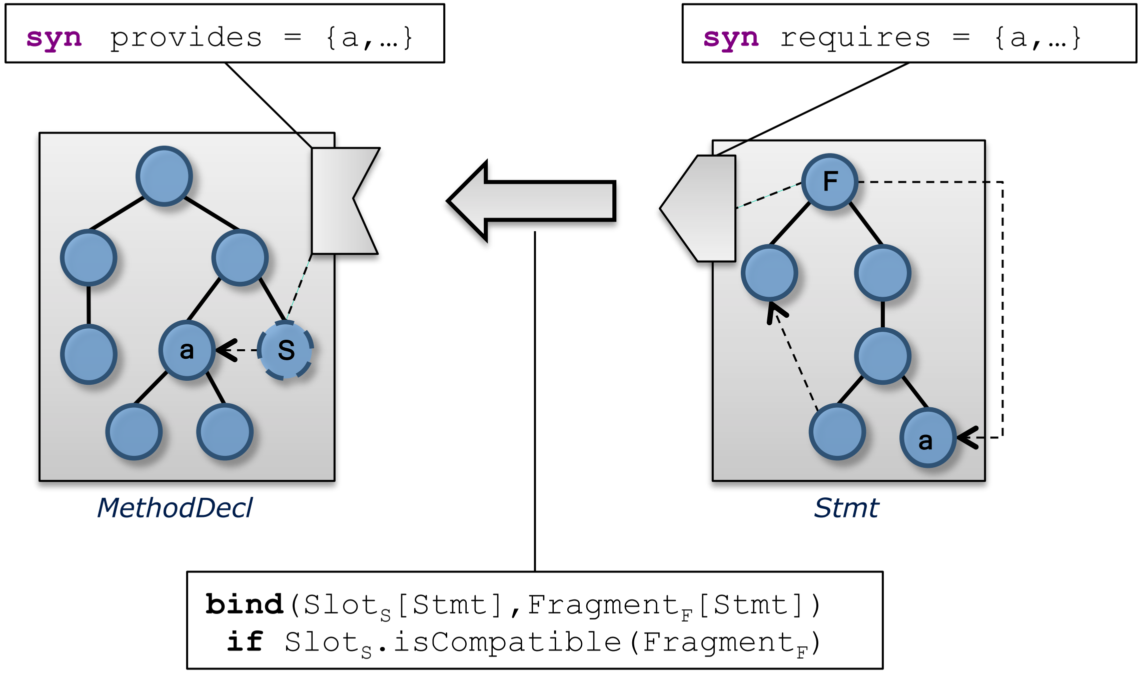
Fragment assertions are (automatically) derived static properties of a given (code) fragment.

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

Fragment Contracts

Example: Def-Use Relation



syn provides = {a,…}

syn requires = {a,…}

bind(Slot$_S$[Stmt],Fragment$_F$[Stmt])
if Slot$_S$.isCompatible(Fragment$_F$)

Java Case Study

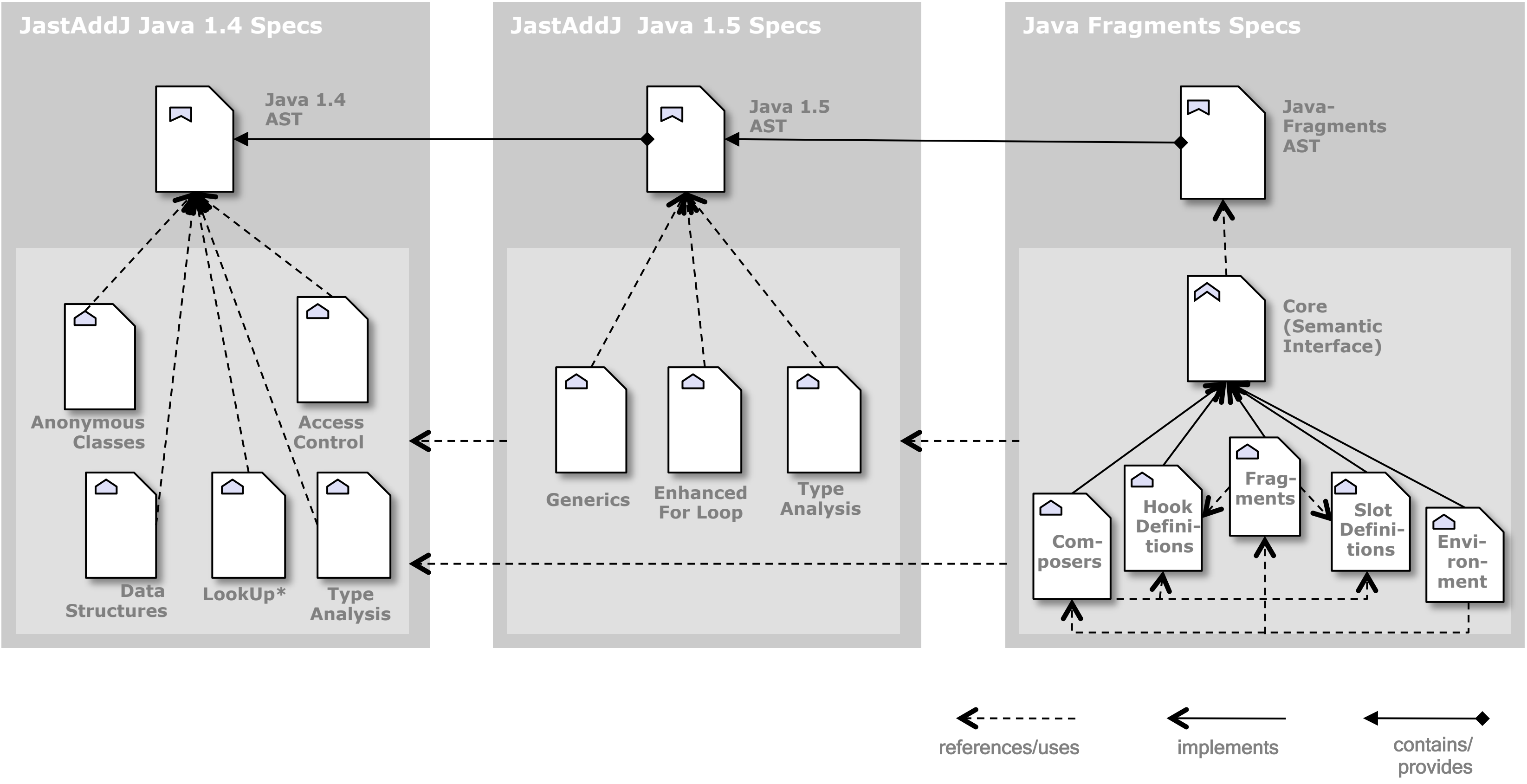
- Java-Fragments based on the RAG tool **JastAdd2** and the **JastAddJ** extensible Java compiler by Hedin/Ekman

- 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

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.«

Fragment Contracts -- Benefits

- Error detection: Contracts are checked before composition → problematic composition steps can be detected
  - ❑ Alternatively a compiler could afterwards find it via some tracelinks
- Composition control: If a contract is not fulfilled at the beginning of the composition, it might still be fulfilled later.
- Efficiency: Caching mechanisms of AGs can make the approach more efficient than a complete re-evaluation/re-compilation
- Expressivity: contract conditions can contain more information than just the information derived from the fragments (fragment assertions vs. contracts).
  - ❑ Example: access restriction to a certain variable (assertion: provided={a,b,…,z}, required={a,b,…,z}, condition: fits(A) if A.required ⊆ S.provided and not b ϵ required)
- Fragment selection/conditional composition: select fragment components fitting to a certain contract or assertion

Background and Motivation

Invasive Software Composition (ISC) [Aßmann 03,Henriksson 09,Johannes 10]

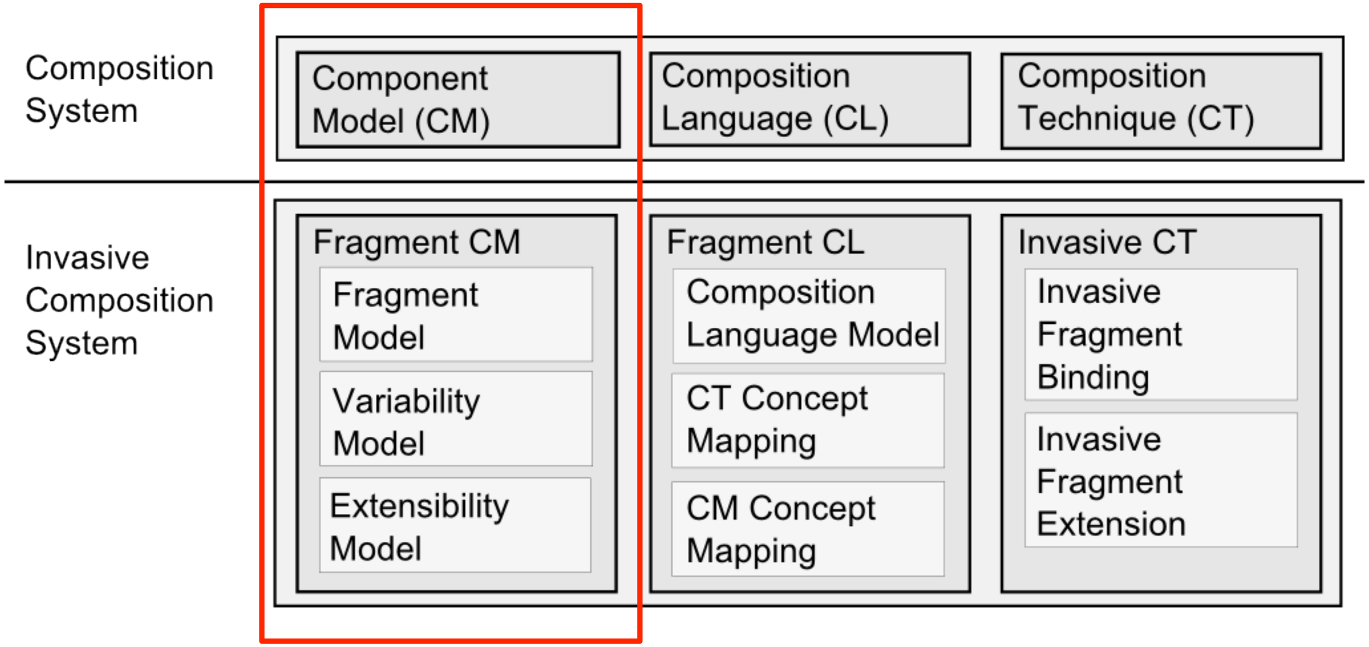
- **ISC** is an approach for fragment-based composition systems
  - *fragment* = partial or under-specified piece of a program or model

- **ISC** is *syntax-safe* according to the grammar or language metamodel.

- Typical applications as add-ons to existing languages:
  - Syntax-safe code generators
  - Syntax-safe pre-processors
  - Model composition
  - Aspect-oriented programming

Background and Motivation

Ingredients of Fragment Composition Systems



Fragment Model

- Extension of the fragment language grammar
  - ❑ Import the language constructs which should be fragment component types (e.g. Methods, Statements, Expressions)
  - ❑ Introduce a new root concept (Environment)
  - ❑ For each fragment component type introduce a corresponding fragment nonterminal

- Example:

```
import MethodDecl, Stmt;
Environment ::= Fragments ;
Fragments ::= Fragment Fragments | Fragment ;
Fragment ::= MethodDeclFragment | StmtFragment ;
MethodDeclFragment ::= <name> MethodDecl ;
StmtFragment ::= <name> Stmt ;
```

Variability Model

- Extending the RAG Spec of the fragment language
  - ❑ Add a synthesised attribute *isSlot* to all language concepts
  - ❑ A *slot-condition* determines if a node in the AST will be a slot, e.g.:
    - an empty method hedged like *"[[" decSlot "]]"*
    - a dedicated language concept like *StmtSlot*
  - ❑ Specify a *slots* collection attribute to make slots available to the composition system

```
syn bool Ni.isSlot ;
fun Ni.isSlot :=
    true, if Ni-slot-condition
    false, otherwise ;

syn Node* Ni.slots := {Slots of all children} ∪ {Ni|Ni.isSlot =
    true} ;
```

RAG-based Fragment Component Models

Extensibility Model

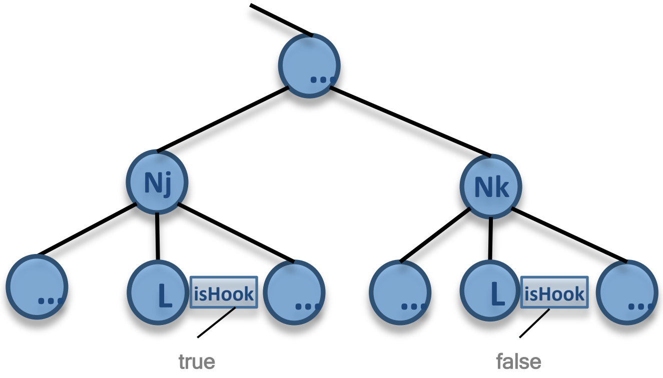
- Further extending the RAG Spec of the fragment language
  - ❑ Add an inherited attribute *isHook* to all (list like) language concepts *L*
    - No physical representation → depend on context
  - ❑ Add a semantic equation for each context of *L*
  - ❑ Add a hooks collection attribute

```
inh bool Ni.isHook ;
syn Node* Ni.hooks := {hooks of all children} ∪ {Ni|Ni.isHook =
    true} ;
```

- Example Contexts:

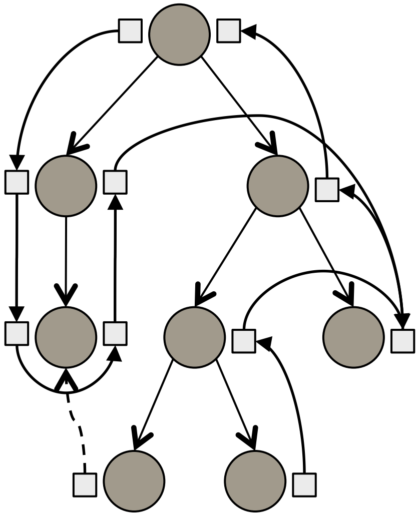
```
Nj ::= ... L ... ;
Nk ::= ... L ... ;
```

```
fun Nj.L.isHook := true ;
fun Nk.L.isHook := false ;
```



Background and Motivation

Kinds of Attributes in Attribute Grammars



- **inherited attributes** (inh): top-down data flow and transformation
- **synthesised attributes** (syn): bottom-up data flow and transformation
- **collection attribute s**(coll): aggregation of values distributed over the AST
- **reference attributes**: computation of reference edges to existing AST nodes

Example: Template-based Code Skeletons in Java

- **Parallel Map Skeleton**

```
ExecutorService executor = Executors.newFixedThreadPool([[WorkerSlot]]);
List<Future<[[ResultTypeSlot]]>> futures = new
LinkedList<Future<[[ResultTypeSlot]]>>();
final List<[[ResultTypeSlot]]> results = new LinkedList<[[ResultTypeSlot]]>();

while([[ExpressionSlot]]){
   //hook
   futures.add(executor.submit(
   new Callable<[[ResultTypeSlot]]>(){
   public ResultTypeSlot call() throws Exception {
      [[ResultTypeSlot]] result = [[CallExpressionSlot]];
      synchronized (results) {
         results.add(result);
      }
      return result;
    }}));}
executor.shutdown();
while(!executor.isTerminated()){//wait}
```

- **Single Threaded Map Skeleton**

```
List<[[ResultTypeSlot]]> results = new LinkedList<[[ResultTypeSlot]]>();
while([[ExpressionSlot]]){
   //hook
   results.add([[CallExpressionSlot]]);
}
```

Slots (code parameters) =
{WorkerSlot, ResultTypeSlot, ExpressionSlot, CallExpressionSlot}

Hooks (code extension points) =
{statements, while4.statements, …}

Fragment/Template Contracts =
{ WorkerSlot *requires* JavaExpression that returns an int,

CallExpressionSlot *requires* JavaExpression of the same type as ResultType (or subtype) }

Example: Expanded Code

- **Parallel Map Skeleton**

```
ExecutorService executor = Executors.newFixedThreadPool(4);
List<Future<Map<String, Integer>>> futures = new LinkedList<Future<Map<String, Integer>>>();
final List<Map<String, Integer>> results = new LinkedList<Map<String, Integer>>();

while(keys.hasNext()){
   //hook
   final text = keys.next();
   futures.add(executor.submit(
   new Callable<Map<String, Integer>>>(){
   public ResultTypeSlot call() throws Exception {
      Map<String, Integer>> result = map(text);
      synchronized (results) {
         results.add(result);
      }
      return result;
    }}));}
executor.shutdown();
while(!executor.isTerminated()){//wait}
```

- **Single Threaded Map Skeleton**

```
List<Map<String, Integer>> results = new LinkedList<Map<String, Integer>>();
while(keys.hasNext()){
   //hook
   final text = keys.next();
   results.add(map(text));
}
```

Dispatcher

If(nCPU>=2 and nData>32k) **:** **use** parallel variant

If(nCPU=1 or nData<=32k) **:** **use** simple variant

Slot Definitions

```
aspect NonterminalSlots{

    eq TypeVariableSlot.IsSlot() = true;
    eq TypeVariableSlot.getChild(int i).isInSlot() = IsSlot();
    eq TypeVariableSlot.SlotName() = IsSlot()?extract(getSlotName(),"[[","]]"):"";
    eq TypeVariableSlot.compatibleFragmentType() = TypeVariable.class;

    eq ExprSlot.IsSlot() = true;
    eq ExprSlot.getChild(int i).isInSlot() = IsSlot();
    eq ExprSlot.SlotName() = IsSlot()?extract(getSlotName(),"[[","]]"):"";
    eq ExprSlot.compatibleFragmentType() = Expr.class;

    eq StmtSlot.IsSlot() = true;
    eq StmtSlot.getChild(int i).isInSlot() = IsSlot();
    eq StmtSlot.SlotName() = IsSlot()?extract(getSlotName(),"[[","]]"):"";
    eq StmtSlot.compatibleFragmentType() = Stmt.class;

    eq TypeAccess.IsSlot() = isHedged(getID(),"[[","]]");
    eq TypeAccess.getChild(int i).isInSlot() = IsSlot();
    eq TypeAccess.SlotName() = IsSlot()?extract(getID(),"[[","]]"):"";
    eq TypeAccess.compatibleFragmentType() = IsSlot()?Access.class:BottomFragmentType.class;

    eq MethodDecl.IsSlot() = name().endsWith("Slot");
    eq MethodDecl.getChild(int i).isInSlot() = IsSlot();
    eq MethodDecl.SlotName() = IsSlot()?name().substring(0,name().length()-4):"";

    eq Literal.IsSlot() = isHedged(getLITERAL(),"[[","]]");
    eq Literal.getChild(int i).isInSlot() = IsSlot();
    eq Literal.SlotName() = IsSlot()?extract(getLITERAL(),"[[","]]"):"";
    eq Literal.compatibleFragmentType() = IsSlot()?Expr.class:BottomFragmentType.class;
}
```