# Concrete Syntax for Objects

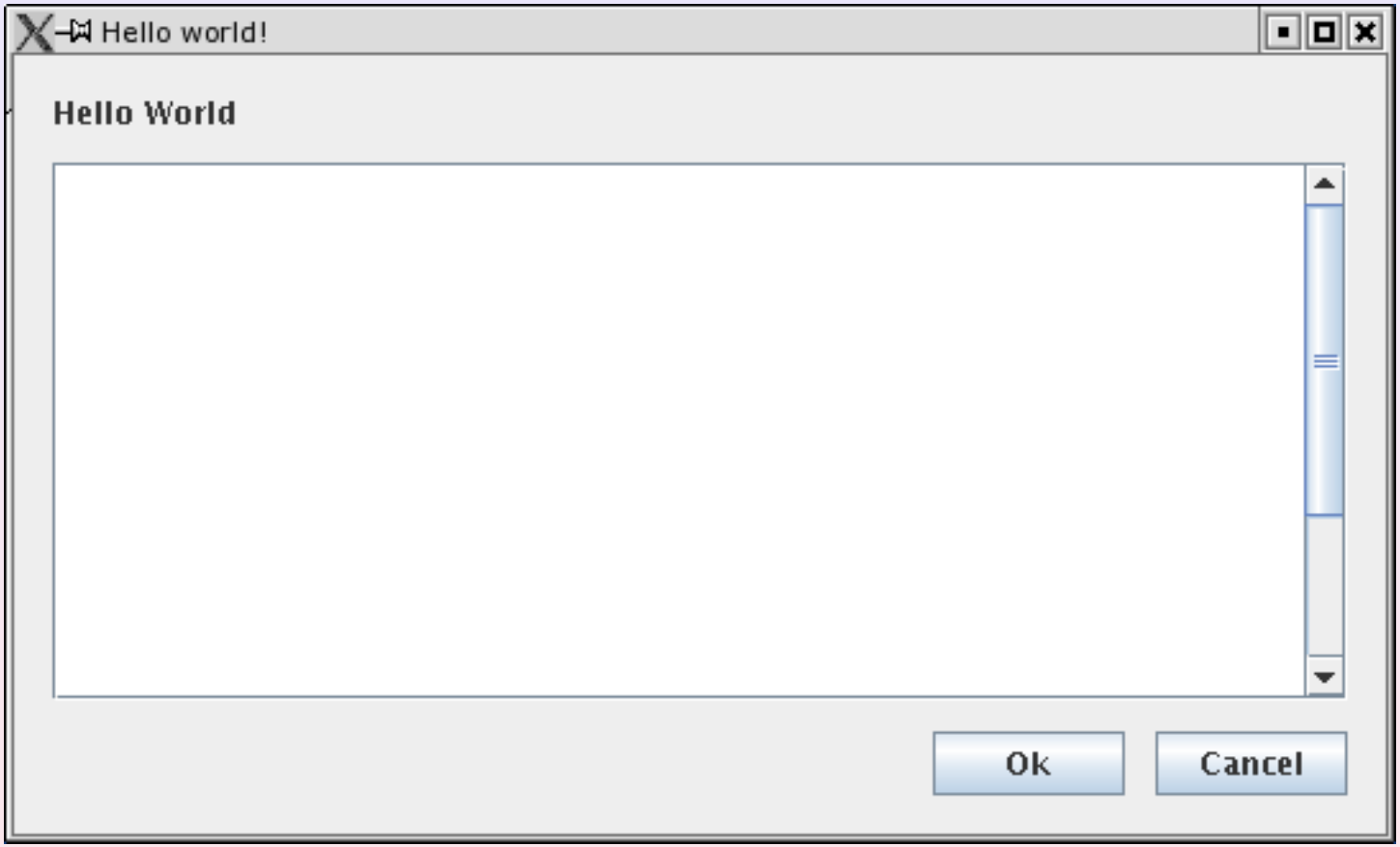
## Domain-Specific Language Embedding and Assimilation without Restrictions

Martin Bravenboer  Eelco Visser

Institute of Information & Computing Sciences
Utrecht University,
The Netherlands

October 28, 2004

# Example 1: Implement a GUI



# Example 1: Implement a GUI using Java/Swing

```
public class HelloWorld {
  public static void main(String[] ps) {

    JTextArea text = new JTextArea(20,40);

    JPanel panel = new JPanel(new BorderLayout(12,12));
    panel.add(BorderLayout.NORTH , new JLabel("Hello World"));
    panel.add(BorderLayout.CENTER , new JScrollPane(text));

    JPanel south = new JPanel(new BorderLayout(12,12));
    JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));
    buttons.add(new JButton("Ok"));
    buttons.add(new JButton("Cancel"));

    south.add(BorderLayout.EAST, buttons);
    panel.add(BorderLayout.SOUTH, south);

    ...
```

# Example 1: Implement a GUI using Java/Swing

```
public class HelloWorld {
  public static void main(String[] ps) {

    JTextArea text = new JTextArea(20,40);

    JPanel panel = new JPanel(new BorderLayout(12,12));
    panel.add(BorderLayout.NORTH , new JLabel("Hello World"));
    panel.add(BorderLayout.CENTER , new JScrollPane(text));

    JPanel south = new JPanel(new BorderLayout(12,12));
    JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));
    buttons.add(new JButton("Ok"));
    buttons.add(new JButton("Cancel"));

    south.add(BorderLayout.EAST, buttons);
    panel.add(BorderLayout.SOUTH, south);

    ...
```

Does not correspond to hierarchical structure of the user-interface.

# Example 1: Implement a GUI using Java/Swing

```
public class HelloWorld {
  public static void main(String[] ps) {

    JTextArea text = new JTextArea(20,40);

    JPanel panel = new JPanel(new BorderLayout(12,12));
    panel.add(BorderLayout.NORTH , new JLabel("Hello World"));
    panel.add(BorderLayout.CENTER , new JScrollPane(text));

    JPanel south = new JPanel(new BorderLayout(12,12));
    JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));
    buttons.add(new JButton("Ok"));
    buttons.add(new JButton("Cancel"));

    south.add(BorderLayout.EAST, buttons);
    panel.add(BorderLayout.SOUTH, south);

    ...
```

Does not correspond to hierarchical structure of the user-interface.

# Example 1: Implement a GUI using Java/Swing

```
public class HelloWorld {
  public static void main(String[] ps) {

    JTextArea text = new JTextArea(20,40);

    JPanel panel = new JPanel(new BorderLayout(12,12));
    panel.add(BorderLayout.NORTH , new JLabel("Hello World"));
    panel.add(BorderLayout.CENTER , new JScrollPane(text));

    JPanel south = new JPanel(new BorderLayout(12,12));
    JPanel buttons = new JPanel(new GridLayout(1, 2, 12, 12));
    buttons.add(new JButton("Ok"));
    buttons.add(new JButton("Cancel"));

    south.add(BorderLayout.EAST, buttons);
    panel.add(BorderLayout.SOUTH, south);

    ...
```

Does not correspond to hierarchical structure of the user-interface.

Analysis of user-interface structure is impossible or difficult.

# Concrete Syntax for Objects

## Domain abstraction in general-purpose languages

- Semantic domain abstraction
  - Designed for extensibility and reuse
- No syntactic domain abstraction
  - Only generic syntax of method invocations
  - No domain-specific notation and composition

# Concrete Syntax for Objects

## Domain abstraction in general-purpose languages

- Semantic domain abstraction
  - Designed for extensibility and reuse
- No syntactic domain abstraction
  - Only generic syntax of method invocations
  - No domain-specific notation and composition

## Concrete syntax for domain abstractions

- Semantic domain abstraction
- Syntactic domain abstraction

The MetaBorg Method:

1. *Embedding* of domain-specific language
2. *Assimilation* of embedded domain code

# Example 1: Implement a GUI using Concrete Syntax

```
public class HelloWorld {
  public static void main(String[] ps) {

    JPanel panel = panel of border layout {
      north = label "Hello World"

      center = scrollpane of textarea {
        rows    = 20
        columns = 40
      }

      south = panel of border layout {
        east = panel of grid layout {
          row = {
            button "Ok"
            button "Cancel"
          }
        }
      }
    };

    ...
```

# Example 1: Implement a GUI using Concrete Syntax

Syntax reflects the hierarchical structure of the user-interface.

```
public class HelloWorld {
  public static void main(String[] ps) {

    JPanel panel = panel of border layout {
      north = label "Hello World"

      center = scrollpane of textarea {
        rows    = 20
        columns = 40
      }

      south = panel of border layout {
        east = panel of grid layout {
          row = {
            button "Ok"
            button "Cancel"
          }
        }
      }
    };

    ...
```

# Example 1: Implement a GUI using Concrete Syntax

```
public class HelloWorld {
  public static void main(String[] ps) {

    JPanel panel = panel of border layout {
      north = label "Hello World"

      center = scrollpane of textarea {
        rows    = 20
        columns = 40
      }

      south = panel of border layout {
        east = panel of grid layout {
          row = {
            button "Ok"
            button "Cancel"
          }
        }
      }
    };

    ...
```

Syntax reflects the hierarchical structure of the user-interface.

The interaction between the domain-specific and general-purpose code is seamless.

# Example 2: Code Generation

Suppose we want to generate:

```
if(propertyChangeListeners == null)
  return;

PropertyChangeEvent event =
  new PropertyChangeEvent(this, fieldName, oldValue, newValue);

for(int c=0; c < propertyChangeListeners.size(); c++) {
  ((PropertyChangeListener)
      propertyChangeListeners.elementAt(c)).propertyChange(event);
}
```

Parameterized by the name of the listeners variable.

(Fragment generated by Castor)

# Example 2: Code Generation using Strings

```
String vName = "propertyChangeListeners";

jsc.add("if (");
jsc.append(vName);
jsc.append(" == null) return;");

jsc.add("PropertyChangeEvent event = new ");
jsc.append("PropertyChangeEvent");
jsc.append("(this, fieldName, oldValue, newValue);");

jsc.add("for (int i = 0; i < ");
jsc.append(vName);
jsc.append(".size(); i++) {");
jsc.indent();
jsc.add("((PropertyChangeListener) ");
jsc.append(vName);
jsc.append(".elementAt(i)).");
jsc.append("propertyChange(event);");
jsc.unindent();
jsc.add("}");
```

# Example 2: Code Generation using Strings

```
String vName = "propertyChangeListeners";

jsc.add("if (");
jsc.append(vName);
jsc.append(" == null) return;");

jsc.add("PropertyChangeEvent event = new ");
jsc.append("PropertyChangeEvent");
jsc.append("(this, fieldName, oldValue, newValue);");

jsc.add("for (int i = 0; i < ");
jsc.append(vName);
jsc.append(".size(); i++) {");
jsc.indent();
jsc.add("((PropertyChangeListener) ");
jsc.append(vName);
jsc.append(".elementAt(i)).");
jsc.append("propertyChange(event);");
jsc.unindent();
jsc.add("}");
```

Uses the Java syntax: the syntax of the domain.

# Example 2: Code Generation using Strings

```
String vName = "propertyChangeListeners";

jsc.add("if (");
jsc.append(vName);
jsc.append(" == null) return;");

jsc.add("PropertyChangeEvent event = new ");
jsc.append("PropertyChangeEvent");
jsc.append("(this, fieldName, oldValue, new

jsc.add("for (int i = 0; i < ");
jsc.append(vName);
jsc.append(".size(); i++) {");
jsc.indent();
jsc.add("((PropertyChangeListener) ");
jsc.append(vName);
jsc.append(".elementAt(i)).");
jsc.append("propertyChange(event);");
jsc.unindent();
jsc.add("}");
```

Uses the Java syntax: the syntax of the domain.

No syntactic checks of the generated code.

Escaping to the meta language is difficult.

Code generator tries to do some pretty printing.

Further processing of the code is impossible.

# Example 2: Code Generation using Abstract Syntax Trees

```
VariableDeclarationFragment fragment =
    _ast.newVariableDeclarationFragment();
fragment.setName(_ast.newSimpleName("event"));
ClassInstanceCreation newi = _ast.newClassInstanceCreation();
newi.setType(_ast.newSimpleType(
    _ast.newSimpleName("PropertyChangeEvent")));
List args = newi.arguments();
args.add(_ast.newThisExpression());
args.add(_ast.newSimpleName("fieldName"));
args.add(_ast.newSimpleName("oldValue"));
args.add(_ast.newSimpleName("newValue"));
fragment.setInitializer(newi);
VariableDeclarationStatement vardec =
    _ast.newVariableDeclarationStatement(fragment);
vardec.setType(_ast.newSimpleType(
    _ast.newSimpleName("PropertyChangeEvent")));
```

# Example 2: Code Generation using Abstract Syntax Trees

Extremely verbose and unclear: 90 lines of code!

```
VariableDecl
    _ast.newVariableDeclarationFragment();
fragment.setName(_ast.newSimpleName("event"
ClassInstanceCreation newi = _ast.newClassI
newi.setType(_ast.newSimpleType(
    _ast.newSimpleName("PropertyChangeEvent")));
List args = newi.arguments();
args.add(_ast.newThisExpression());
args.add(_ast.newSimpleName("fieldName"));
args.add(_ast.newSimpleName("oldValue"));
args.add(_ast.newSimpleName("newValue"));
fragment.setInitializer(newi);
VariableDeclarationStatement vardec =
    _ast.newVariableDeclarationStatement(fragment);
vardec.setType(_ast.newSimpleType(
    _ast.newSimpleName("PropertyChangeEvent")));
```

Does not correspond to the structure of the code to be generated.

# Example 2: Code Generation using Abstract Syntax Trees

Extremely verbose and unclear: 90 lines of code!

```
VariableDecl                                    :
    _ast.newVariableDeclarationFragment();
fragment.setName(_ast.newSimpleName("event"
ClassInstanceCreation newi = _ast.newClassI
newi.setType(_ast.newSimpleType(
    _ast.newSimpleName("PropertyChangeEvent")));
List args = newi.arguments();
args.add(_ast.newThisExpression());
args.add(_ast.newSimpleName("fieldName"));
args.add(_ast.newSimpleName("oldValue"));
args.add(_ast.newSimpleName("newValue"));
fragment.setInitializer(newi);
VariableDeclarationStatement vardec =
    _ast.newVariableDeclarationStatement
vardec.setType(_ast.newSimpleType(
                                    ngeEvent")));
```

Does not correspond to the structure of the code to be generated.

Code is syntactically checked by host language compiler and further processing is possible.

Don't worry about the layout.

# Example 2: Code Generation using Concrete Syntax

```
String x = "propertyChangeListeners";

List<Statement> stms = |[
  if(x == null)
    return;

  PropertyChangeEvent event =
    new PropertyChangeEvent(this, fieldName, oldValue, newValue);

  for(int c=0; c < x.size(); c++) {
    ((PropertyChangeListener)
      x.elementAt(c)).propertyChange(event);
  }
]|;
```

# Example 2: Code Generation using Concrete Syntax

```
String x = "propertyChangeListeners";

List<Statement> stms = |[
  if(x == null)
    return;

  PropertyChangeEvent event =
    new PropertyChangeEvent(this, fieldName, oldValue, newValue);

  for(int c=0; c < x.size(); c++) {
    ((PropertyChangeListener)
      x.elementAt(c)).propertyChange(event);
  }
]|;
```

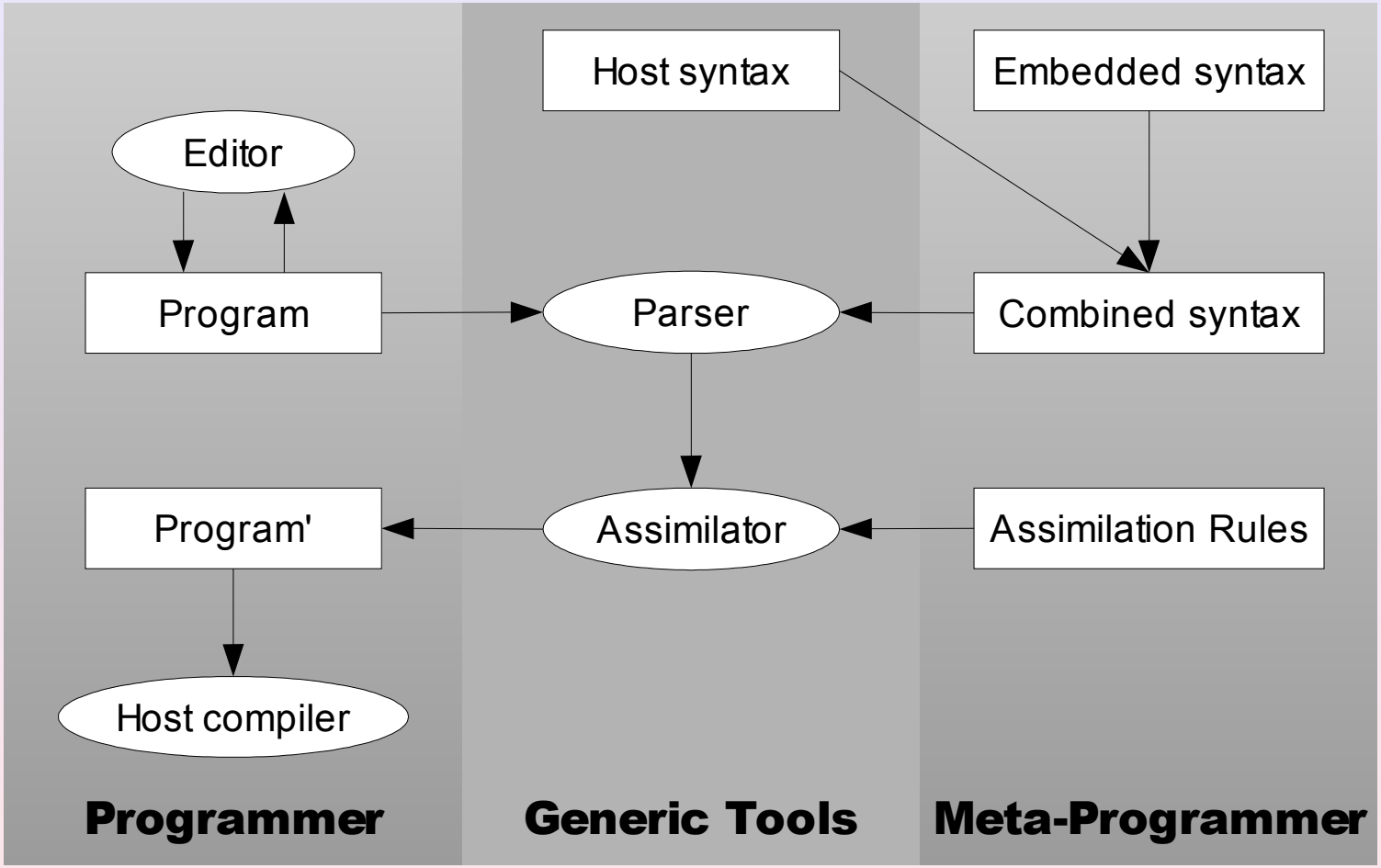
Uses the syntax of the domain: Java.

Syntax of the generated code is checked and further processing is possible.

Separate pretty-printer: don't worry about the layout.

Support for interaction between the generated code and the meta language.

# Architecture of the MetaBorg Method



# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# Characteristics of the MetaBorg Method

1. **Syntactic**
   - Syntax of embedded code checked at compile-time
2. **No restrictions on syntax**
   - Arbitrary context-free languages
   - Embed languages with different lexical syntax
3. **Not specific to single host language**
   - Embed domain syntax in any host language
4. **Interaction with host language**
   - Weave embedded code in host language
5. **Combination of extensions**
   - Embed multiple languages
6. **No restrictions on assimilation**
   - Context-sensitive, global
   - Optimization, semantic checks

# MetaBorg Applied: Java/Pairs

Syntactic domain abstraction

```
(Integer, String) t = (1, "Hello world!");
```

API: Semantic domain abstraction

```
public class Pair<F, S> {
  public Pair(F first, S second) ...
  public static <F1, S1> Pair<F1, S1> construct(F1 f, S1 s)

  public F getFirst()  ...
  public void setFirst(F value)  ...
}
```

After assimilation

```
Pair<Integer, String> t = Pair.construct(1 , "Hello world!");
```

# Realizing Java/Pairs

Embed syntax for Pairs in Java

```
module Java-Pair imports Java-15
exports
  context-free syntax
    "(" Expr "," Expr ")" -> Expr {cons("NewPair")}
    "(" Type "," Type ")" -> Type {cons("PairType")}
```

Assimilate Pairs to Pair API

```
module Java-Pair-Assimilate imports Java-Pair
rules
  AssimilatePair :
    expr |[ (e1, e2) ]| -> expr |[ Pair.construct(e1, e2) ]|

  AssimilatePair :
    type |[ (t1, t2) ]| -> type |[ Pair<t1, t2> ]|
```

# Realizing Java/Pairs

Embed syntax for Pairs in Java

```
module Java-Pair imports Java-15
exports
  context-free syntax
    "(" Expr "," Expr ")" -> Expr {cons("NewPair")}
    "(" Type "," Type ")" -> Type {cons("PairType")}
```

Assimilate Pairs to Pair API

```
module Java-Pair-Assimilate imports Jav
rules
  AssimilatePair :
    expr |[ (e1, e2) ]| -> expr |[ Pair.construct(e1, e2) ]|

  AssimilatePair :
    type |[ (t1, t2) ]| -> type |[ Pair<t1, t2> ]|
```

Assimilation rules use concrete syntax for Java and Pairs as well!

# MetaBorg Applied: Java/Swul

## Syntactical domain abstraction

```
"panel" "of" Layout -> Component {cons("Panel")}
"button" String     -> Component {cons("ButtonText")}
```

## Embedding of domain specific language

```
Component -> Expr      {cons("ToExpr")}
Expr      -> Component {cons("FromExpr")}
```

## Assimilation rules

```
Swulc-Component :
  swul |[ button e ]| -> expr |[ new JButton(e) ]|

Swulc-Layout :
  swul |[ grid layout {ps*} ]| -> expr |[ new GridLayout(i,j) ]|
  where <nr-of-rows> ps* => i
      ; <nr-of-columns> ps* => j
```

# MetaBorg Applied: JavaJava

## Embed Java syntax in Java

```
"e" [0-9]*      -> Expr          {prefer}
"e" [0-9]* "*" -> {Expr ","}*   {prefer}
"type" "|[" Type "]|" -> MetaExpr {cons("ToMetaExpr")}
```

## Assimilation rules for Eclipse JDT Core API

```
Assimilate(r) :
  type |[ double ]| -> |[ ast.newPrimitiveType(PrimitiveType.DOUBLE) ]|

Assimilate(r) :
  |[ e; ]| -> |[ ast.newExpressionStatement(~e: <r> e) ]|

Assimilate(r) :
  |[ y(e*) ]| -> |[
    {| MethodInvocation x = ast.newMethodInvocation();
       x.setName(ast.newSimpleName("~y"));
       bstm* | x |}
  ]|
  where <newname> "inv" => x
      ; <ExplodeArgs(r | x)> e* => bstm*
```

# MetaBorg Applied: JavaJava

## Embed Java syntax in Java

```
"e" [0-9]*      -> Expr          {prefer}
"e" [0-9]* "*" -> {Expr ","}*   {prefer}
"type" "|[" Type "]|" -> MetaExpr {cons("ToMetaExpr")}
```

## Assimilation rules for Eclipse JDT Core API

```
Assimilate(r) :
  type |[ double ]| -> |[ ast.newPrimitiveType(PrimitiveType.DOUBLE) ]|

Assimilate(r) :
  |[ e; ]| -> |[ ast.newExpressionStatement(~e: <r> e) ]|

Assimilate(r) :
  |[ y(e*) ]| -> |[
    {| MethodInvocation x = ast.newMeth
       x.setName(ast.newSimpleName("~y'
       bstm* | x |}
  ]|
  where <newname> "inv" => x
      ; <ExplodeArgs(r | x)> e* => bstm*
```

To make the implementation of assimilation rules easier, declarations and statements are allowed in expressions.

# MetaBorg Foundations

MetaBorg embeddings are relatively easy to implement. Why?

- SDF
  - Modular syntax definition
  - Defines lexical and context-free syntax
  - Declarative disambiguation
  - Allows ambiguities
- SGLR
  - Scannerless Generalized LR parsing
  - Lexical analysis is context-sensitive
- Stratego
  - Strategies and rewrite rules
  - Meta-programming with concrete syntax

*All available and proven technology!*

# Scope of MetaBorg

- Meta programming
  - *Code generation* (run-time)
  - Annotation processing
- *Graphical user interfaces*
- Embedded query languages
  - XPath, XQuery, SQL, JDOQL
- Language processing
  - Context-free grammars
  - Regular expressions
- *XML processing*
- Concurrency abstractions
- ...

*Available as prototype in JavaBorg*

# Conclusion

**MetaBorg: Concrete Syntax for Domain Abstractions**

- *Embedding* of domain-specific language
- *Assimilation* of embedded domain code

Embedded domain-specific languages . . .

- make code more readable.
- encourage a better style of programming.
- future work: integration in compilers, debuggers, refactoring tools, documentation generators, etc.

`http://www.metaborg.org`