

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

June 3, 2004

Outline

Programming with Concrete Syntax

- Graphical User-Interfaces

- XML Processing

- Java Code Generation

Realizing Concrete Syntax for Objects

- Overview of MetaBorg

- Java/Tuples

- Java/Swul

Technical Foundations

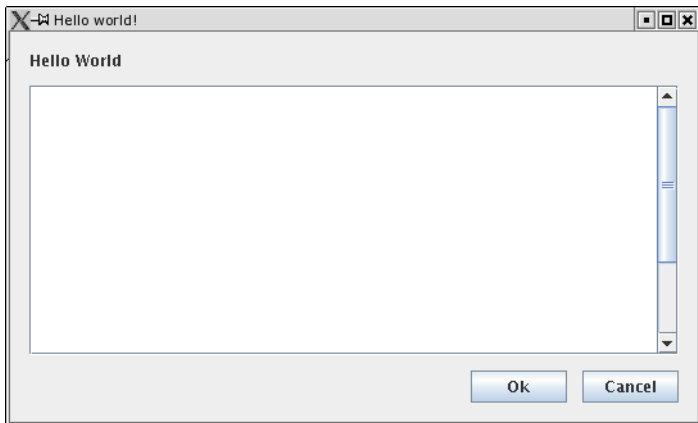
- Modular Syntax Definition

- Scannerless GLR Parsing

- Rewrite Rules

Conclusion

Simple Java/Swing GUI



GUI: Implementation in Java

```

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);
  
```

GUI: Implementation in Java/Swul

```

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"
      }
    }
  }
};
  
```

XML fragment (Cocoon)

```
<html>  
  <body>  
    <p>Some text here</p>  
  </body>  
</html>
```

XML Generation in Java (Cocoon)

```
out.startDocument();
out.startElement("", "html", "html", noAttrs);
out.startElement("", "body", "body", noAttrs);
out.startElement("", "p", "p", noAttrs);
out.characters(text.toCharArray(), 0, text.length());
out.endElement("", "p", "p");
out.endElement("", "body", "body");
out.endElement("", "html", "html");
out.endDocument();
```

XML Generation in Java/XML

```
out.write document %>
  <html>
    <body>
      <p><% text :: cdata %></p>
    </body>
  </html>
<%;
```

Java code (Castor)

```
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);
}
```

Java Generation in Java (Castor)

```

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

Java Generation in JavaJava

```

ATerm stm = bstm [| {
    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);
    }
} |];
    
```

Philosophy of MetaBorg

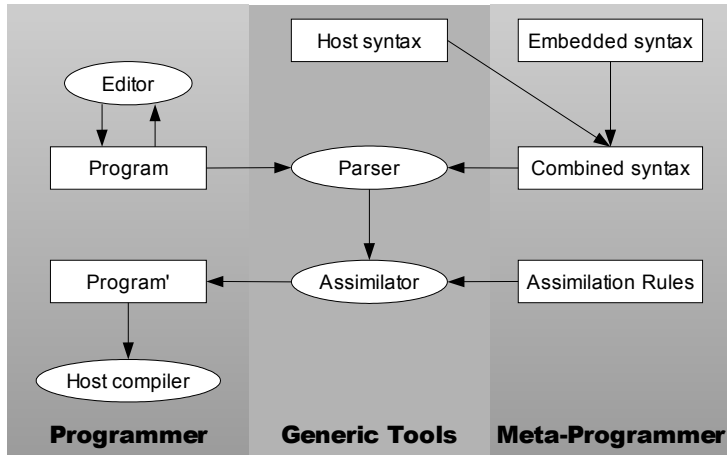
General purpose languages:

1. Designed for extensibility and reuse
⇒ Sufficient *semantic* domain abstraction
2. API: Generic syntax of method invocations
⇒ Poor *syntactic* domain abstractions

MetaBorg: Concrete syntax for domain abstractions

- ▶ *Embedding* of domain-specific language
- ▶ *Assimilating* embedded domain code

Architecture of MetaBorg



Java/Tuples

Syntactic abstraction:

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

API:

```
public class Tuple<F, S> {  
    public Tuple(F first, S second) ...  
    public static <F1, S1> Tuple<F1, S1> construct(F1 f, S1 s) ...  
  
    public F getFirst() ...  
    public void setFirst(F value) ...  
}
```

After assimilation:

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

Realizing Java/Tuples

Embed syntax of Tuples in Java

```

module Java-Tuple imports Generic-Java
exports
  context-free syntax
    "(" Expr "," Expr ")" -> Expr {cons("NewTuple")}
    "(" Type "," Type ")" -> Type {cons("TupleType")}
    
```

Assimilate Tuples to Tuple API

```

module Java-Tuple-Assimilate imports Generic-Java
rules
  AssimilateTuple :
    expr [[ (e1, e2) ]] -> expr [[ Tuple.construct(e1, e2) ]]

  AssimilateType :
    type [[ (t1, t2) ]] -> type [[ Tuple<t1, t2> ]]
    
```

Realizing Java/Tuples

Embed syntax of Tuples in Java

```

module Java-Tuple imports Generic-Java
exports
  context-free syntax
    "(" Expr "," Expr ")" -> Expr {cons("NewTuple")}
    "(" Type "," Type ")" -> Type {cons("TupleType")}
    
```

Assimilate Tuples to Tuple API

```

module Java-Tuple-Assimilate imports Generic-Java
rules
  AssimilateTuple :
    expr [[ (e1, e2) ]] -> expr [[ Tuple.construct(e1, e2) ]]

  AssimilateTuple :
    type [[ (t1, t2) ]] -> type [[ Tuple<t1, t2> ]]
    
```

SDF Syntax Definition of Swul

```
module Swul imports Swul-Layout
exports
```

context-free syntax

```
"panel" "of" Layout      -> Component {cons("Panel")}
"panel" "{" PanelProp* "}" -> Component {cons("Panel")}
"layout" "=" Layout     -> PanelProp {cons("Layout")}
"border" "=" Border     -> PanelProp {cons("Border")}
```

context-free syntax

```
"button" String          -> Component {cons("ButtonText")}
"button" "for" Action     -> Component {cons("Button")}
"button" "{" ButtonProp* "}" -> Component {cons("Button")}
```

context-free syntax

```
Id ":@" Component -> Component {cons("Assign")}
Id ":" Component -> Component {cons("Declare")}
```

lexical syntax

```
[a-zA-Z][a-zA-Z0-9]+ -> Id
```

Syntax Definition of Java/Swul

Embedding Swul in Java:

- ▶ Swul constructs as Java expressions
- ▶ Java expressions as Swul constructs

```

module Java-Swul
imports Java-Prefixed Swul-Prefixed
exports
  context-free syntax
  SwulComponent -> JavaExpr {cons("ToExpr")}
  SwulLayout    -> JavaExpr {cons("ToExpr")}

  JavaExpr      -> SwulBorder {cons("FromExpr")}
  JavaExpr      -> SwulComponent {cons("FromExpr")}
    
```

Assimilation Rules for Java/Swul

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
```

Swulc-AddComponent(|x) :

```
swul |[ south = c ]| -> bstm |[ x.add(BorderLayout.SOUTH, e); ]|
  where <Swulc-Component> c => e
```

Swulc-Component :

```
swul |[ x := c ]| -> expr |[ { | x = e; | x | } ]|
  where <Swulc-Component> c => e
```

Swulc-Component :

```
swul |[ x : c ]| -> expr |[ { | t x = e; | x | } ]|
  where <java-type-of> c => t
        ; <Swulc-Component> c => e
```

Assimilation Rules for Java/Swul

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
```

Swulc-AddComponent(|x) :

```
swul [[ south = c ]] -> bstm [[ x.add(BorderLayout.SOUTH, e); ]]  

    where <Swulc-Component> c => e
```

Swulc-Component :

```
swul [[ x := c ]] -> expr [[ { | x = e; | x | } ]]  

    where <Swulc-Component> c => e
```

Swulc-Component :

```
swul [[ x : c ]] -> expr [[ { | t x = e; | x | } ]]  

    where <java-type-of> c => t  

         ; <Swulc-Component> c => e
```

Why is MetaBorg this Easy?

MetaBorg is based on the right tools for the job.

- ▶ SDF
Modular syntax definition
- ▶ SGLR
Scannerless Generalized LR parsing
- ▶ Stratego
Rewriting rules with concrete syntax

Modular Syntax Definition

Modularity:

- ⇒ requires full class of context-free grammars.
- ⇒ requires disambiguation

SDF: Modular to the core.

- ▶ defines complete (lexical and context-free) syntax
- ▶ allows full class of context-free grammars
- ▶ allows ambiguities
- ▶ declarative disambiguation
- ▶ modules, renamings, parameterization

SDF: Disambiguation

Associativity:

```
Exp "+" Exp -> Exp {left, cons("Plus")}
```

Relative priorities and group associativity:

context-free priorities

```
Exp "." Id "(" {Exp ","}* ")" -> Exp
> {left:
    Exp "/" Exp -> Exp
    Exp "*" Exp -> Exp }
> {left:
    Exp "+" Exp -> Exp
    Exp "-" Exp -> Exp }
```

Parsing with Separate Lexical Analysis

Lexical Analyzer:

- ▶ Input string to sequence of tokens
- ▶ Often generated from regular expressions

JPanel	north	=	panel	of	border	layout	{	north	=
id	kwd	=	kwd	kwd	kwd	kwd	{	kwd	=

Scannerless Parsing

Embedding languages:

- ▶ *Context* of token must be considered
- ▶ Longest-match might be too greedy
- ▶ Separate lexers are not composable.

Solution:

- ▶ no separate lexical analysis: *scannerless*
- ▶ complete definition of syntax
- ▶ optional specification of longest-match:

lexical restrictions

```
Id          -/- [A-Za-z0-9]
IntConst    -/- [0-9]
```

Structured Tokens

A lexer produces tokens: plain sequence of characters.

```
href="http://www.cs.uu.nl/staff/<% name %>.html"
```

SGLR parses this without any further processing to:

```
Attribute(  
  QName(None, "href")  
  , DoubleQuoted(  
    [ Literal("http://www.cs.uu.nl/staff/")  
      , Literal(FromMetaExpr(ExprName(Id("name"))))  
      , Literal(".html")  
    ]  
  )  
)
```

Assimilation: Rewrite Rules

Stratego's rewrite rules:

- ▶ Implement *basic assimilation* steps
- ▶ Application is controlled by a *strategy*.
- ▶ Many assimilations can reuse *generic traversal strategies*.
- ▶ Dynamically defined rewrite rules for *context-sensitive* assimilation.
- ▶ Expressed in concrete syntax for embedded domain-specific language and host language: MetaBorg!

Distinguishing Characteristics

- ▶ Syntactic
- ▶ No restrictions on syntax definition
- ▶ Not restricted to a single host language
- ▶ Interaction with host language
- ▶ No restrictions on assimilation

Conclusion: Overview

MetaBorg method

- ▶ Domain abstractions at syntactic level
- ▶ Embedding of domain-specific languages
- ▶ Assimilation of domain code

MetaBorg application prospect

- ▶ The growth of a language is restricted by its general application area.
- ⇒ Extension for domain-specific applications.

Excited?

Related courses

- ▶ Seminar: Software Generation and Configuration
- ▶ Course: Program Transformation

Master projects

- ▶ Linguistic reflection
- ▶ Strategic Java
- ▶ ...

More information

- ▶ <http://www.metaborg.org> (paper and code)
- ▶ <http://www.syntax-definition.org>
- ▶ <http://www.stratego-language.org>