The METABORG *Method*

SIGOURE Benoit

LRDE – EPITA Research and Development Laboratory

1er mars 2006

Table of contents

1. *Overview*
2. *The world of* METABORG
3. *How to use* METABORG
4. *Case Study : SWUL*
5. *Case Study : JavaJava*

Table of contents

1. Overview
2. The world of MetaBorg
3. How to use MetaBorg
4. Case Study : SWUL
5. Case Study : JavaJava

Domain-Specific Language Embedding and Assimilation without Restrictions

- METABORG is a method for providing concrete syntax (using SDF and Stratego) for domain abstraction.

Domain-Specific Language Embedding and Assimilation without Restrictions

- METABORG is a method for providing concrete syntax (using SDF and Stratego) for domain abstraction.
- What is concrete syntax ? What is domain abstraction ?
- Why use METABORG ?

Domain-Specific Language Embedding and Assimilation without Restrictions

- METABORG is a method for providing concrete syntax (using SDF and Stratego) for domain abstraction.
- What is concrete syntax ? What is domain abstraction ?
- Why use METABORG ?

- Concrete syntax : representation of a program in a given programming language (as seen in the source file, with its layout, parentheses, etc.)

Domain-Specific Language Embedding and Assimilation without Restrictions

- METABORG is a method for providing concrete syntax (using SDF and Stratego) for domain abstraction.
- What is concrete syntax ? What is domain abstraction ?
- Why use METABORG ?

- Concrete syntax : representation of a program in a given programming language (as seen in the source file, with its layout, parentheses, etc.)
- Abstract syntax : expresses the structure of the concrete syntax independantly of its physical representation.

Domain-Specific Language Embedding and Assimilation without Restrictions

- METABORG is a method for providing concrete syntax (using SDF and Stratego) for domain abstraction.
- What is concrete syntax ? What is domain abstraction ?
- Why use METABORG ?

- Concrete syntax : representation of a program in a given programming language (as seen in the source file, with its layout, parentheses, etc.)
- Abstract syntax : expresses the structure of the concrete syntax independantly of its physical representation.
- Domain abstraction : in opposition with domain-specific.
  ⇒ For example : XML is specific for data exchange.

Some of the typical things you might want to do with METABORG.

- Using a domain-specific language within another language ;
- Extend a language ;
- Promote an API to the language level (eg, SWUL) ;
- Meta Programming (eg, JavaJava, Transformers).

Why use METABORG *: Some examples*

- Using a domain-specific language within another language.

XML Generation in Java (with 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();
```

⇒ Embed XML in Java using METABORG and write the following code instead :

JavaXML

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

Why use METABORG *: Some examples*

- Extend a language.

Tuples in Java

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

⇒ Embed syntax of Tuples in Java and you can write the following code :

JavaTuple

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

Table of contents

1 Overview

2 The world of MetaBorg

3 How to use MetaBorg

4 Case Study : SWUL

5 Case Study : JavaJava

The world of METABORG *: The XT Bundle*

METABORG is just a particular pattern of usage of the tools in the Stratego/XT Bundle.

METABORG is just a particular pattern of usage of the tools in the Stratego/XT Bundle.

The Stratego/XT Bundle contains :

- ATerm : Annotated Term ;

METABORG is just a particular pattern of usage of the tools in the Stratego/XT Bundle.

The Stratego/XT Bundle contains :

- ATerm : Annotated Term ;
- SDF : Syntax Definition Formalism ;

The world of METABORG *: The XT Bundle*

METABORG is just a particular pattern of usage of the tools in the Stratego/XT Bundle.

The Stratego/XT Bundle contains :

- ATerm : Annotated Term ;
- SDF : Syntax Definition Formalism ;
- SGLR Parser : Scannerless Generalized LR Parser ;

METABORG is just a particular pattern of usage of the tools in the Stratego/XT Bundle.

The Stratego/XT Bundle contains :

- ATerm : Annotated Term ;
- SDF : Syntax Definition Formalism ;
- SGLR Parser : Scannerless Generalized LR Parser ;
- Stratego : Language for Program Transformations.

ATerm (Annotated Term)

- Format used to represent abstract syntax in a structured way. Supports lists, tuples, integers, strings, constructors and annotations.

Concrete syntax (Java)

```
4 + var.foo(5, 1)
```

⇒

Abstract syntax (Aterm)

```
Plus(
  Int("4")
, Call(
    Var("var")
  , "foo"
  , [Int("5"), Int("1")]
  )
)
```

- Maximal sharing of redundant data for size-efficiant representations of big inputs.
- ATerms can be freely annotated.
- Stratego is manipulating ATerms (matching and rewriting them).

SDF (Syntax Definition Formalism)

- Concise support of context-free languages ;
- Allows separate disambiguation filters instead of having to hack the syntax definition ;
- Modular approach (syntax definition can be split in independant modules) ;
- Used by SGLR to parse the input source code (concrete syntax) into an AST of ATerms ;
- Does not match the input stream in a greedy way.

SGLR (Scannerless Generalized LR Parser)

- Generalized : produces a parse forest of all possible parse trees if the grammar is ambiguous ;
- Scannerless : considers the context of the input stream before deciding what kind of token is formed by the lexeme read.

The importance of Scannerless Parsing

Considere the following input in JavaXML :

```
<a href="http://www.<% s %>.org">Link</a>
```

When the parser reaches the variable s, it automatically interprets it as Java code using the contextual information provided by the layout "<%"

Table of contents

1. *Overview*
2. *The world of* METABORG
3. *How to use* METABORG
4. *Case Study : SWUL*
5. *Case Study : JavaJava*

The METABORG model works in two steps.

- Embedding the domain-specific language(s) into the host the host language.
  - Each language shall be well defined using SDF ;
  - Care must be taken so that each grammar has its own non-terminal names in order to prevent undesired embeddeding.
- Assimilating the embedded language(s) into the host language.
  - Adding productions to combine the syntaxes of the language(s) in both ways ;
  - host language $\Rightarrow$ assimilated language(s) ;
  - host language $\Leftarrow$ assimilated language(s).

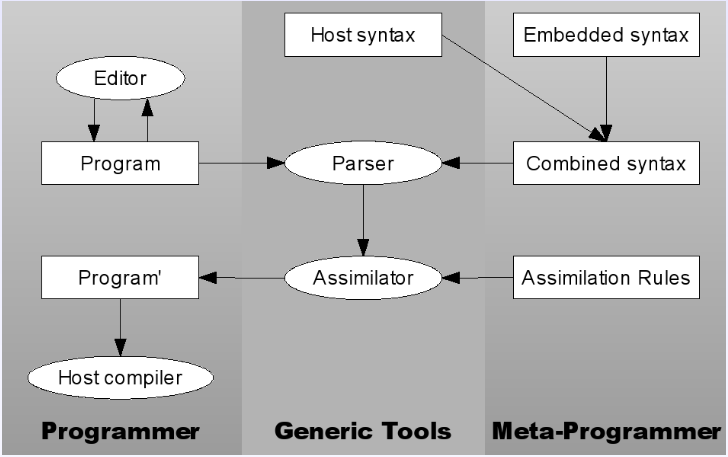

FIG.: The Architecture of METABORG

Concrete application of METABORG *: Java Tuples*

- Let's say we want to extand a language.
- For instance, we might want to add some sugar in Java in order to ease the declaration of Tuples.

Tuples in Java

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

We would rather like to write :

JavaTuple

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

Let's embed the syntax of Tuples in Java

SDF of Tuples in Java

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

Concrete application of METABORG *: Java Tuples*

- Let's say we want to extand a language.
- We want to add some sugar in Java in order to ease the declaration of Tuples.

Let's embed the syntax of Tuples in Java

SDF of Tuples in Java

```
module Java-Tuple imports Generic-Java
exports
  context-free syntax
    "(" Expr "," Expr ")" -> Expr {cons("NewTuple")}
    "(" Type "," Type ")" -> Type {cons("TupleType")}
  variables
    "e" [0-9]*              -> Expr {prefer}
    "t" [0-9]*              -> Type {prefer}
    "expr"                  -> Expr {prefer}
    "type"                  -> Type {prefer}
```

Using the injection of meta-language StrategoTerms into object language Expressions it is possible to distinguish meta-variables from object language identifiers. Thus, in the term `|[ var ~x := ~e]|`, the expressions `~x` and `~e` indicate meta-level terms, and hence x and e are meta-level variables.
NOTE : The prefer attribute ensures that these identifiers are preferred over normal identifiers.

Now let's add a set of rewriting rules which will embed Tuples in Java

JavaTuple

```
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> |]
```

- And that's all you need to write !

Table of contents

1. *Overview*
2. *The world of* METABORG
3. *How to use* METABORG
4. *Case Study : SWUL*
5. *Case Study : JavaJava*

- The SWING library offers an API which is quite hard to use.
- Writing a simple GUI requires many intermediate variables.
- We want to introduce a more specific notation to easily use the library.

Swing User-Interface Language

Using SWING...

```
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);
```

Swing User-Interface Language

Using 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"
       }
     }
   }
};
```

SWUL SDF

```
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

Embedding SWUL in Java

```
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
```

Table of contents

1. *Overview*
2. *The world of* METABORG
3. *How to use* METABORG
4. *Case Study : SWUL*
5. *Case Study : JavaJava*

JavaJava

Extend the principle of SWUL, but instead of using SWUL, introduce Java in Java. This enables very easy constructions to generate Java code using Java programs.

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);
    }
  }]|;
```

Questions ?