MetaBorg

An Approach for Domain-Specific Language Embedding

Martin Bravenboer Eelco Visser

TU Delft

Delft University of Technology

November 17, 2006

The quest for abstraction

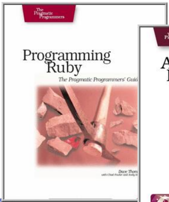
Libraries, languages, frameworks

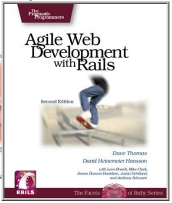
- Query-languages (SQL, XPath, XQuery, OQL, JDOQL, . . .)
- Command-languages (Shell, PowerShell)
- XML processing (SAX, DOM, . . .)
- User-interface (Swing, SWT, WinForms, . . .)
- Application frameworks (EJB, Hibernate, Struts, EJB, Rails)

Pro's and con's

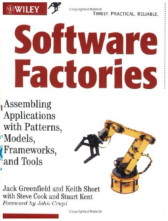
- Very useful domain abstractions
- Not the right abstractions at syntactic level
  - Notation, domain composition, structure, symbolic integration

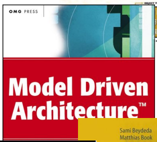
Solution camps: extreme abstraction

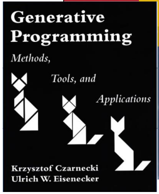


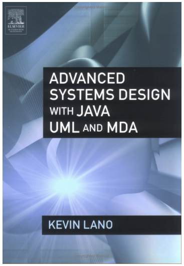

The Pragmatic Programmers
Agile Web Development with Rails
Second Edition
Dave Thomas
David Heinemeier Hansson
The Facets of Ruby Series

MDA Explained
The Model Driven Architecture: Practice and Promise
Anneke Kleppe
Jos Warmer
Wim Bast
Foreword by Andrew Watson


WILEY
Timely. Practical. Reliable.
Software Factories
Assembling Applications with Patterns, Models, Frameworks, and Tools
Jack Greenfield and Keith Short
with Steve Cook and Stuart Kent
Foreword by John Crupi







Domain-specific languages

YACC, ANTLR, JavaCC, SDF, XQuery, SQL, XPath, OCL, OQL, HQL, JDOQL, EJBQL, XSLT, SVG, MathML, sed, grep, Make, spreadsheets, regular expressions, automaton, . . .

Challenges

- Development cost
- Scope, domain-specificity ⇔ general-purpose
- Disruptive in the development process
- Tracing abstractions (performance, debugging)

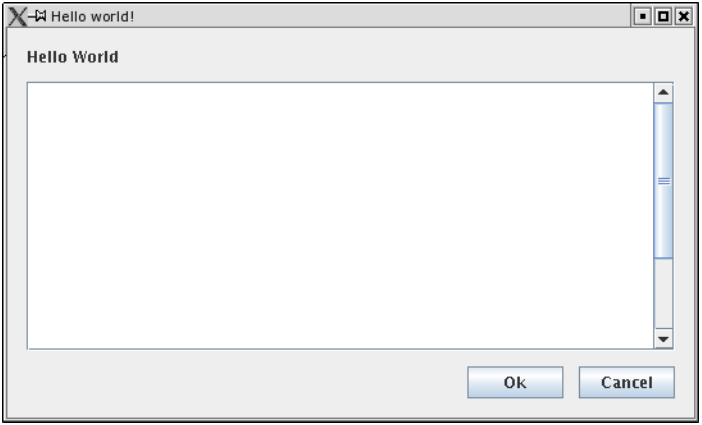
Proposed solution: the MetaBorg method

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

MetaBorg metaphor

MetaBorg provides generic technology for allowing a host language (collective) to incorporate and **assimilate external domains** (cultures) in order to strengthen itself. The ease of implementing embeddings makes resistance futile.

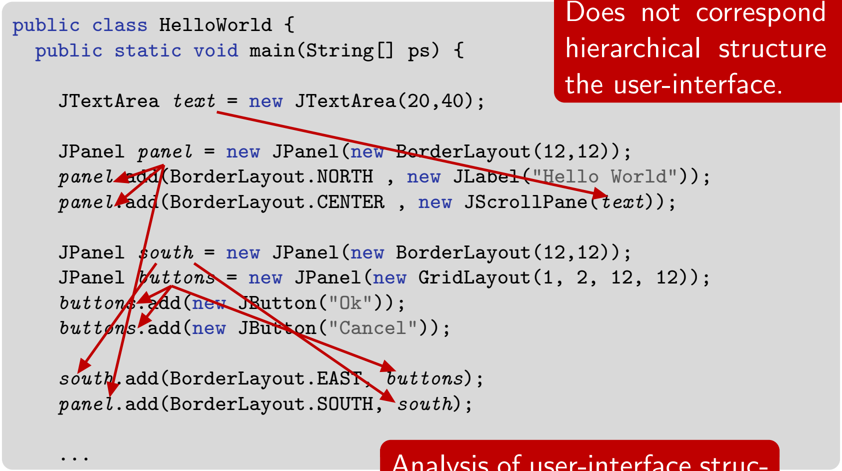
Example 1: Implement a GUI



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.

Analysis of user-interface structure is impossible or difficult.

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

```
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 1: Implement a GUI using concrete syntax

```
menu item { text = "New" accelerator = ctrl-N }
```

```
JMenuItem newfile = new JMenuItem("New");
newfile.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, 2));
```

```
menu item {
  text = "Exit"
  action event = { System.exit(0); }
}
```

```
JMenuItem_0 = new JMenuItem();
JMenuItem_0.setText("Exit");
JMenuItem_0.addActionListener(
  EventHandler.create(..., ClassHandler_0, "ActionListener_0", ""));

public static class ClassHandler_0 {
  public void ActionListener_0(ActionEvent event) { System.exit(0); }
}
```

Example 2: SQL in Java

```
String userName = ...;
String password = ...;
String query = "SELECT * FROM users "
  + "WHERE name = '" + userName + "'"
  + "AND password = '" + password + "'";

if(executeQuery(query).size() == 0) ...
```

```
String userName = ...;
String password = ...;
SQL q = <| SELECT id FROM users
           WHERE name = ${userName}
           AND password = ${password} |>;

if (executeQuery(q.toString()).size() == 0) ...
```

Example 3: SQL in PHP

```
$username = $_GET['username'];
$q = "SELECT * FROM users "
   . "WHERE username = '" . $username . "'";

executeSQL($q);
```

```
$username = $_GET['username'];
$q = <| SELECT * FROM users
        WHERE username = ${$username} |>;

executeSQL($q->toString());
```

Example 4: Shell in PHP

```
$command = "svn cat \"file name\" -r" . $rev;
system($command);
```

```
$command = <| svn cat "file name" -r${$rev} |>;
system($command->toString());
```

Example 5: Regular expressions in Java

```
Pattern ipline = Pattern.compile(
  "( ( [0-1]?\\d{1,2} \\. ) | ( 2[0-4]\\d \\. ) | ( 25[0-5] \\. ) ){3}
   ( ( [0-1]?\\d{1,2}     ) | ( 2[0-4]\\d     ) | ( 25[0-5]     ) )");

if(ipline.matcher(input).matches()) {
  System.out.println("Input is an ip-number.");
} else {
  System.out.println("Input is NOT an ip-number.");
}
```

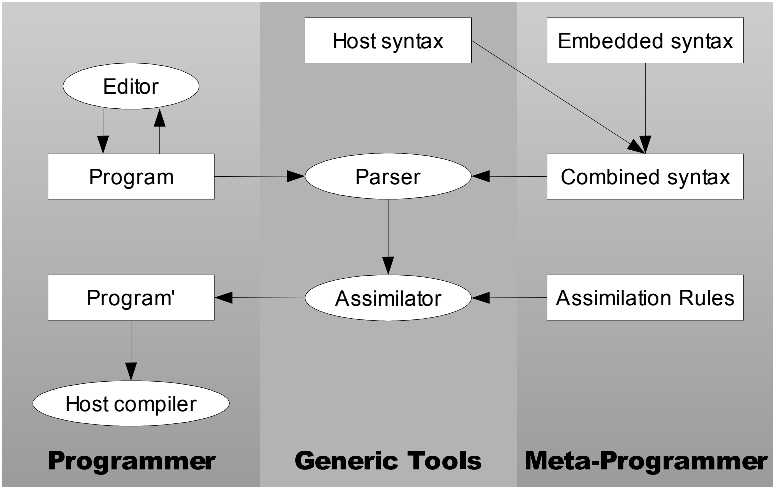
```
regex ipline = [/
    ( ( [0-1]?\d{1,2} \. ) | ( 2[0-4]\d \. ) | ( 25[0-5] \. ) ){3}
    ( ( [0-1]?\d{1,2}    ) | ( 2[0-4]\d    ) | ( 25[0-5]    ) )
  /];

if( input ~? ipline ) {
  System.out.println("Input is an ip-number.");
} else {
  System.out.println("Input is NOT an ip-number.");
}
```

Example 7: String rewriting

```
String input = ...

regex body = [/ <body[^>]*?> .* </body> /]
regex amp = [/ & /] -> [/ &amp; /];
regex lt  = [/ < /] -> [/ &lt;  /];
regex gt  = [/ > /] -> [/ &gt;  /];
input ~= one(body <~> all(amp <+ lt <+ gt))
```

```
conversion string -> CharString {
  prefix "\'";
  suffix "\'";

  escape {
    [\'] -> "\'\'";
  }
}
```

Architecture of the MetaBorg method



MetaBorg technology: Stratego/XT

Assimilation

- Rewrite rules

  Code generation in small, declarative steps
- Rewrite strategies

  Control application of rewrite rules
- Concrete syntax

  Code generation using familiar syntax

Syntax embedding

- Modular syntax definition

  Composition of languages
- Scannerless generalized-LR parsing

  Elegantly deals with syntax embedding issues

MetaBorg results: publications

- **Concrete Syntax for Objects**
  OOPSLA'04 conference
- **Generalized Type-Based Disambiguation**
  GPCE'05 conference
- **MetaBorg in Action**
  GTTSE'05 Journal
- **Transformations for Abstractions**
  SCAM'05 workshop keynote
- **Syntax Definition for AspectJ**
  OOPSLA'06 conference
- **Preventing Injection Attacks with Syntax Embeddings**
  Submitted to ICSE'07 conference

MetaBorg: open questions

- Scope of the method
  - Application domains
  - Application frameworks
- Integration of abstractions at different levels
  - Easy at same level of abstraction
  - Interaction and references between extensions
- Development environment integration
  - Symbolic integration IDE
  - Refactoring, debugging, documentation generators
- Finding the right abstractions
  - Evolution of the embedded DSLs

Strong impression that these are more general issues