Rewriting Object Models (With Cycles and Nested Collections)

A Model-Based Metaprogramming Problem

Markus Lepper[1] **Baltasar Trancón y Widemann**[1,2]

[1]semantics GmbH, Berlin, DE

[2]Ilmenau University of Technology, Ilmenau, DE

6th ISoLA
2014-10-11

From the Track Abstract

> *In the last years, model-based software development received more and more attraction.*

> *Compilers nearly perform the same task as model-based code generators.* [...]
> *However, the technology used for the implementation of compilers is rather different.*

- Wasted opportunities on either side?
- Missing link?
  - Evidence?

Meta- (Generative) Programming

- Generation of source code artifacts
- Integral part of development process
- Extension of programmer's freedom of expression

Text
- **Homoiconic** macros, templates
- **Heteroiconic** external DSLs

API
- **Construction** deeply embedded DSLs
- **Action** shallowly embedded DSLs

GUI wizards

The MetaTools Approach

Declarative Partial Programming

Declarative
- high-level stateless descriptions
- formal semantics
- practically exploitable properties

Partial
- flexible trade-off with manual coding
- clean interface (in either direction)
- embrace host module & type system
- non-invasive (in either direction)
- robust to changes (on either side)
- 'smart separate compilation' [XSE'01, ASE 10]

Classic Example: Visitor Style Pattern

- High-end control pattern in ‘gang-of-four’ book
- Reification of data structure traversal

- Base class defines one method per node type
- Dynamic dispatch $\Longrightarrow$ type matching
- Default behavior: traversal of successors

- User overrides methods to modify type-local behavior

Action extract information
Delegation modify traversal order

The MetaTools Suite

- Extensive collection of metaprogramming tools
- Host technologies: Java, XML
- Both internal & external DSLs
- Cross-bootstrapping
- Mid-scale applications
  - compilers (some first-to-market)
  - document management systems
  - well-typed XSLT processor

http://www.bandm.eu/metatools/

Proprietary Example: μMod

- Data model generator
- Very concise textual description
- Formal semantics
- Best of both worlds: OO & ADT

OO
- inheritance (incl. constructors)
- arbitrary graphs
- collection-valued attributes
- visitors

ADT
- **null**-safety
- immutable types
- stable deep equality & hash
- pattern matching
- pretty-printing combinators

Example μMod Model

```
MODEL Sig
VISITOR 0 Visitor
VISITOR 0 Rewriter IS REWRITER
TOPLEVEL CLASS
    Statement ABSTRACT
    | Assignment
        left     SEQ Variable                                   ! V 0/0
        right    Expression                                     ! V 0/1
    | Block
        stmts    SEQ Statement                                  ! V 0/0
    Expression ABSTRACT
    | Reference
        var      OPT Variable                                   ! V 0/0
    Variable ALGEBRAIC
        id       int
    Statistics
        vars     Statement -> bool -> SET Variable              ! V 0/0
```

Visitor Support in μMod

- Annotate model attributes with traversal plans
- Generate visitor base code
- Optimization potential [ICMT'11]
  - generated code known to have no effect
  - user overrides detectable by reflection
  - combined static & dynamic analysis
  - switch off improductive traversal per subclass

Generated Visitor Code

```
package Sig;
abstract class Visitor {                              // V 0/*
  ⋮
  void action(Block b) {
    for (Statement s : block.get_stmts())             // V */0
      match(s);
  }
  void action(Assignment a) {
    match(a.get_left());                              // V */0
    match(a.get_right());                             // V */1
  }
}
```

User-Defined Visitor Code

```
Program copyPropagation(Program prog) {
  final Map<Variable, Variable> copies = new HashMap<>();
  new Visitor() {
    @Override void action(Assignment a) {
      // match pattern "Assignment({x}, Reference(y))" against "a"
      if (/* success */)
        copies.put(x, y);
      super.action(a);                                  // top down
    }
  }.match(prog);
  // ... see below ...
}
```

The Rewriter Pattern

- Visitors are asymmetric:

  Declarative type-based matching (method resolution)
  Imperative reaction by side effect

- Extend pattern by transparent rewriting
  - avoid in-place mutation ***but***
  - support arbitrary reaction code
  - type-safe ***even***
  - **with collection-valued attributes**
  - traversal-based control flow ***but***
  - support sharing & cycles

- Denotational semantics for pure subset [CALCO'13]
- Competitor in model transformation tool contest [TTC'11,SCP 85]

Rewriter Workflow

- Rules applied locally to shallow clones
- Changes propagate backwards

  Change clone retained
  No change clone disposed; original shared

- User code overrides local behavior
  - shallow mutation of clone
  - substitution by different subgraph
  - modification of traversal order
  - ...
- Optional features
  - global cache for maximal sharing
  - transparent cycle detection & handling [PhD'07]

User-Defined Rewriter Code

```
Program copyPropagation(Program prog) {
  final  Map<Variable, Variable> copies = new HashMap<>();
  // ...  see above ...
  return (Program)new Rewriter() {
    @Override void rewriteFields (Variable  v) {
      if  (copies.containsKey(v))
        substitute (copies.get(v));                    // propagate
    }
    @Override void rewriteFields (Assignment a) {
      super.rewriteFields (a) ;                        // bottom up
      // match pattern "Assignment({x}, Reference(y))"  against  "a"
      if  (/* success */ && x.equals(y))               // now redundant?
        substitute_multi ()  ;                         // eliminate
    }
  }.rewrite (prog);
}
```

Rewriting Collections

- Structured adjacency
- Compositional attribute type constructs
  *****, **SEQ**, **SET**, **MAP**/->, **REL**/<->
- Transparent transitive rewriting?
  - mathematical intuition vs. imperative implementation
  - conflicts from convergent rewrites
  - one-to-many rewrites
  - principle of least surprise
- Simplification: one **REL** to rule them all

  to straight-forward for all constructs

  fro fails for lhs conflicts in **MAP**

  - good candidate for PoLS
  - static inference & dynamic checks

The Joy of Maps

Given $\{\{a \mapsto b\} \mapsto c, \{a \mapsto d\} \mapsto e\}$

Rewrite $b \rightsquigarrow d$

Conflict (unless $c \downarrow e$)

- Imperative ‘implementation’ (`put`) violates PoLS
- Dynamic checks
  - unpleasant safety holes
  - potentially costly (**MAP**s of **MAP**s)
- Static inference
  - rewriting is constant $\Rightarrow$ injective on primitives
  - some sound upward propagation rules
- Direct implementation (in principle, backdoor)

Summary

- Declarative swords are two-edged
  - theoretically & practically superior properties
  - limited power & interoperability with legacy
- Pragmatic trade-off
  - pure ‘sandboxes’ often not affordable
  - programming discipline must be leveraged
  - generate as much as possible
- Make things work together
  - inheritance as robust interface
  - static & dynamic safety nets

Self-Bibliography I

- Jakumeit, E. et al. (2014). "A survey and comparison of transformation tools based on the transformation tool contest". In: *Sci. Comput. Program.* 85 , pp. 41–99. DOI: 10.1016/j.scico.2013.10.009. URL: http://dx.doi.org/10.1016/j.scico.2013.10.009.
- Lepper, M. and B. Trancón y Widemann (2011). "Optimization of Visitor Performance by Reflection-Based Analysis". In: *Theory and Practice of Model Transformations, ICMT 2011*. Ed. by J. Cabot and E. Visser. Vol. 6707. LNCS. Springer, pp. 15–30. DOI: 10.1007/978-3-642-21732-6_2.
- Lepper, M. and B. Trancón y Widemann (2011). "Solving the TTC 2011 Compiler Optimization Task with metatools". In: *Proceedings 5th Transformation Tool Contest (TTC 2011)*. Ed. by P. van Gorp, S. Mazanek, and L. Rose. Vol. 74. Electr. Proc. Theor. Comp. Sci. Pp. 70–115. DOI: 10.4204/EPTCS.74.9.

Self-Bibliography II

- Lepper, M., B. Trancón y Widemann, and J. Wieland (2001). "Automated Generation of Abstract Syntax Trees Represented as Typed DOM XML". In: *Proceedings International Workshop on XML Technologies and Software Engineering (XSE 2001)*. Ed. by W. Emmerich, C. Mascolo, and A. Finkelstein. University College London.
- Trancón y Widemann, B. (2007). "Strikte Verfahren zyklischer Berechnung". Dissertation. Technische Universität Berlin.
- Trancón y Widemann, B. and M. Lepper (2014). "Towards a (Co)Algebraic Semantics for the Object-Oriented Rewriter Pattern". In: *Proceedings 5th Algebra and Coalgebra in Computer Science (CALCO 2013), Early Ideas Workshop*. Ed. by M. Seisenberger, pp. 49–62. URL: `http://www-compsci.swan.ac.uk/~csmona/CALCO-EI-Selected-Papers.pdf`.

Self-Bibliography III

Trancón y Widemann, B., M. Lepper, and J. Wieland (2003). "Automatic Construction of XML-Based Tools, Seen as Meta-Programming". In: *Automated Software Engineering* 10.1 , pp. 22–38. DOI: 10.1023/A:1021864801049.