Reflection and Context

Marcus Denker
marcus.denker@inria.fr
http://rmod.lille.inria.fr

© Marcus Denker
Roadmap

I. Sub-Method Structural Reflection
II. Partial Behavioral Reflection
III. Meta Context
Smalltalk

> Smalltalk has support for reflection

> Structural reflection
  — Classes / methods are objects
  — Can be changed at runtime

> Behavioral reflection
  — Current execution reified (thisContext)
  — #doesNotUnderstand / MethodWrappers
Can we do better?

> Structural Reflection stops at method level
  — Bytecode in the CompiledMethod: Numbers
  — Text: Just a String, needs to be compiled

> Behavior hard coded in the Virtual Machine
  — Message Sending
  — Variable Access

> Both structural and behavioral reflection is limited
  — We should do better!
Structural Reflection

> Structure modeled as objects
  
  — e.g. Classes, methods
  — Causally connected

> Uses:
  
  — Development environments
  — Language extensions and experiments
Methods and Reflection

> **Method are Objects**
  — e.g in Smalltalk

> **No high-level model for sub-method elements**
  — Message sends
  — Assignments
  — Variable access

> **Structural reflection stops at the granularity of methods**
Sub-Method Reflection

> Many tools work on sub method level
  — Profiler, Refactoring Tool, Debugger, Type Checker

> Communication between tools needed
  — Example: Code coverage

> All tools use different representations
  — Tools are harder to build
  — Communication not possible
Existing Method Representations

Existing representations for Methods

- Text
- Bytecode
- AST
Requirements

- Causal Connection
- Abstraction Level
- Extensibility
- Persistency
- Size and Performance
> Low level abstraction
  — String of characters

> Not causally connected
  — Need to call compiler
Bytecode

- **Low level abstraction**
  - Array of Integers

- **Missing extensibility**
  - e.g. for tools

- **Mix of base- and meta-level code**
  - Problems with synthesized code when changing code
  - Examples: AOP point-cut residues, reflection hooks
Abstract Syntax Tree

> Not causally connected
  — Need to call compiler

> Not extensible
  — Fixed set of codes, no way to store meta data

> Not persistent
  — Generated by compiler from text, never stored
Solution: Reflective Methods

- Annotated, persistent AST
- Bytecode generated on demand and cached
Persephone

> Implementation of Reflective Methods for Squeak 3.9

> Smalltalk compiler generates Reflective Methods
  — Translated to bytecode on demand

> Open Compiler: Plugins
  — Called before code generation
  — Transform a copy of the AST
Requirements revisited

> Abstraction Level   OK

> Causal Connection   OK

> Extensibility       OK

> Persistency         OK

> Size and Performance OK
Annotations

> Source visible annotations
   — extended Smalltalk syntax

(9 raisedTo: 10000) <:evaluateAtCompiletime:>

> Source invisible annotations
   — Reflective API
   — Can reference any object

> Every node can be annotated

> Semantics: Compiler Plugins
Example: Pluggable Type-System

Example for textual annotations

```plaintext
bitFromBoolean: aBoolean <::type: Boolean :>
^ (aBoolean ifTrue: [1] ifFalse: [0]) <::type: Integer :>
```

Optional, pluggable type-system

Types stored as annotations in the Reflective Methods
## Memory

<table>
<thead>
<tr>
<th></th>
<th>number of classes</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squeak 3.9</td>
<td>2040</td>
<td>15.7 MB</td>
</tr>
<tr>
<td>Persephone no reflective methods</td>
<td>2224</td>
<td>20 MB</td>
</tr>
<tr>
<td>Persephone reflective methods</td>
<td>2224</td>
<td>123 MB</td>
</tr>
</tbody>
</table>
Roadmap

- I. Sub-Method Structural Reflection
- II. Partial Behavioral Reflection
- III. Meta Context
Behavioral Reflection

> Reflect on the execution
  — method execution
  — message sending, variable access

> In Smalltalk
  — No model of execution below method body
  — message sending / variable access hard coded by VM
  — #doesNotUnderstand / MethodWrappers

> Reflective capabilities of Smalltalk should be improved!
**MetaclassTalk**

> Extends the Smalltalk metaclass model
  — Similar to CLOS MOP

> **Metaclass defines**
  — message lookup
  — access to instance variables

> **Problems:**
  — Reflection only controllable at class boundaries
  — No fine-grained selection (e.g. single operations)
  — Protocol between base and meta level is fixed
Reflex: Partial Behavioral Reflection

> Hooksets: collection of operation occurrences

> Links
  — Bind hooksets to meta-objects
  — Define protocol between base and meta

> Goals
  — Highly selective reification
  — Flexible meta-level engineering
    – Protocol specification
    – Cross-cutting hooksets

Tanter, OOPSLA03
Example: Profiler

> Operation:
  — Method execution (around)

> Hookset:
  — All execution operations in a package

> Meta-object:
  — A profiling tool
Reflex for Squeak

> **Partial Behavioral Reflection pioneered in Java**
  — Code transformation at load time
  — Not unanticipated (it’s Java...)

> **Geppetto: Partial Behavioral Reflection for Smalltalk**
  — For Squeak 3.9 with Bytecode transformation
Problems

> Annotation performance
  — Decompile bytecode

> Execution performance
  — Preambles for stack manipulation

> Low-level representation
  — ifTrue:ifFalse:
  — Blocks
  — Global variables
Links as Annotations

> Links can be annotations on the AST
Behavioral Reflection: Flexible

> Very Flexible
Behavioral Reflection: CLOS

Meta-class MOP (CLOS)
Behavioral Reflection: AOP

> Aspects
Behavioral Reflection: Tracer

> Tracer
Properties

> **Very fast annotations**
  — No decompile!

> **On-the-fly code generation**
  — Only code executed gets generated

> **Generated code is fast**
  — Better then working on bytecode level
Demo

> Show Bounce Demo
Reflectivity

> Prototype implementation in Squeak
  
  — Sub-Method Structure
  — Partial Behavioral Reflection

> Download:

http://scg.unibe.ch/Research/Reflectivity
> Not yet...

— Now we can do it for real!
  – *Engineering vs. Research*...
Roadmap

I. Sub-Method Structural Reflection
II. Partial Behavioral Reflection
III. Meta Context
Behavioral Reflection: Flexible

Let’s use it!
Problem: Recursion

> Behavioral reflection cannot be applied to the whole system

— System classes
— Meta-objects
Example: Beeper

> Call the Beeper from OrderedCollection>>#add

beepLink := Link new metaObject: Beeper.
beepLink selector: #beep.

(OrderedCollection>>#add:) methodNode link: beepLink.
Meta-object Call Recursion

Base Level  Meta Object  Meta Object

#add: send

#beep send  #add: send

#beep send  #add: send

Infinite recursion
Ad-hoc Solutions

- Code duplication
- Adding special tests
Tower of Interpreters

> Smith, 1982
The Real Problem

Representing Meta-Level Execution
The Meta-Context

> Link enables MetaContext
Context-aware Links

> Disable call when already on the meta-level
MetaContext

> Recursion problem solved
Meta-level analysis:
— Trace the tracer
MetaContext

> Recursion problem

> Missing representation of meta-level execution

> Meta-context
  — Solves the recursion problem
  — Enables *meta-level analysis*
Rethinking Reflection

> Meta change “shows through”
  — Introspection shows implementation
  — Recursion and confusion of meta levels

> Reflective change is always global
  — Any change is visible to the whole system
  — No way to batch multiple changes into one atomic operation
Next steps

> Generalize context model:
  — Beyond context as control flow.

> Virtual machine support... to make it practical

> What is the next reflective language kernel?
A lot of open questions...

that's why it is Research...

??????
Questions
License

http://creativecommons.org/licenses/by-sa/2.5/

Attribution-ShareAlike 2.5

You are free:
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.