Reflection

Marcus Denker
denker@iam.unibe.ch

Universität Bern
Roadmap

- Introduction: Reflection
- I. Sub-Method Structural Reflection
- II. Partial Behavioral Reflection
Roadmap

> **Introduction: Reflection**
> I. Sub-Method Structural Reflection
> II. Partial Behavioral Reflection
System

Definition:

A **computational system** is a computer-based system whose purpose is to answer questions and/or support actions about some domain.

Causally Connected

Definition:

A system is said to be causally connected to its domain if the internal structures and the domain they represent are linked in such a way that if one of them changes, this leads to a corresponding effect of the other.

(Patty Maes, OOPSLA 87)
Definition:

A reflective system is a system which incorporates causally connected structures representing (aspects of) itself.

(Patty Maes, OOPSLA 87)
> Introspection
  — Self-representation can be queried

> Intercession
  — Self-representation can be changed

Reflection = Introspection + Intercession
Structure and Behavior

> Structural Reflection
  — Concerned with static structure
  — For example: packages, data-types, procedures

> Behavioral Reflection
  — Concerned with execution
  — For example: procedure execution, assignment, variable read
Tower of Interpreters

- First studied for procedural languages
- David A. Smith: 3Lisp
- Tower-of-Interpreters
- Theoretical. Slow!

<table>
<thead>
<tr>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

User Program running a level 0
Interpreter at level 1
Interpreter at level 2
Interpreter at level 3
Reflection and OOP

> A good match: self-representation build of objects
  - Better then interpreter data-structures

> Language-based reflection
  - Language entities represented as objects
  - Meta-objects describe behavior of base level objects

> Structure: classes/methods are objects
> Behavior: meta-objects define behavior
  - Example: meta-class defines method lookup
Example: Java

- Structural introspection
  - `java.lang.reflect`
  - Query a model of the program (classes, protocols)

- Limited intercession
  - No change of classes

- Limited behavioral reflection
  - Wrappers on objects
  - No way to intercept method calls, variable access
Example: Squeak

> Squeak 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!
Roadmap

> Introduction: Reflection
> I. Sub-Method Structural Reflection
> II. Partial Behavioral Reflection
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

> 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

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

> Optional, pluggable type-system
> Types stored as annotations in the Reflective Methods

© Marcus Denker
## 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><strong>Persephone</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no reflective methods</td>
<td>2224</td>
<td>20 MB</td>
</tr>
<tr>
<td><strong>Persephone</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reflective methods</td>
<td>2224</td>
<td>123 MB</td>
</tr>
</tbody>
</table>
Roadmap

> Introduction: Reflection in Squeak
> I. Sub-Method Structural Reflection
> II. Partial Behavioral Reflection
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
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
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
What’s next...

> Optimize Size of AST Representation
  - Simpler AST
  - AST Compression

> Beyond Text
  - Store only AST (no text)
  - Build text from annotated AST
License

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

**BY:** 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.