Marcus Denker
Software Composition Group
University of Berne - Switzerland
Software Composition Group
University of Berne

> Lead by Oscar Nierstrasz and Stephane Ducasse
> Overall Focus: Software Evolution
> Two parts:
  — Evolution of Existing Systems (Reengineering)
    — Moose, CodeCrawler
  — Language Design for enabling Evolution
    — Traits
    — ClassBoxes

> Forward and Reverse engineering viewpoints
  — We start to see many parallels / cross fertilization
Roadmap

> Reflective Systems
  — Behavioral Reflection
  — Squeak’s Reflective capabilities

> Methods in Squeak
  — Methods as Objects
  — Objects as Methods

> ByteSurgeon and Geppetto
  — Usage
  — Problems

> Beyond text
Reflection

> Object oriented model of the system available inside the system
   — called “Introspection”
   — Java

> Model is *causally connected*
   — Changing this model changes the system
   — called “Intercession”

> Reflection = Introspection + Intercession
Behavioral and Structural

> Structural reflection: changing structure
  — Add / remove classes and methods
  — Add / remove instance variables
  — Change inheritance relationship

> Behavioral reflection: changing behavior
  — What is inheritance?
  — Hook into instance variable stores (e.g. persistence)

> Both are related
  — change of structure changes behavior
Usage: Why Reflection

> Structural reflection
  — Changing systems at runtime
  — Powerful development environments (no edit-compile-run)
  — Analysis (through introspection)

> Behavioral reflection
  — Language experiments
  — Debugging
  — Dynamic analysis (tracing, visualization)
  — New language features (e.g. persistence)
Squeak: A Reflective System

> Squeak: open source Smalltalk
  – Classes and methods are objects
  – Changing these objects changes the system (at runtime)

> API for
  – adding / removing classes + methods
  – adding / removing instance variables
  – changing inheritance relationship
Behavorial Reflection: only by changing methods

There is no API for introspection/intercession of:
- Instance variable access
- Temp variable access
- Message sending
- Message lookup
- Method execution
Structural Reflection enables Behavioral Reflection

> General: Change of structure changes behavior
> We can use the structural reflection API to provide behavioral reflection
  - Methods are objects
  - We can just replace them with our version that does what we want
Behavioral Reflection: Howto?

> Method Wrappers (e.g. used by AspectS)
  — Gives access to before / after of method execution

> Squeak’s Objects-As-Methods
  — we can install any object as a method that implements a simple protocol (#run:with:in)
  — used by ClassBoxes, FacetS
  — reifies method execution

> Transformation of text / AST / Bytecode
ByteSurgeon

> Framework for editing bytecode for Squeak
  — Like Javasist in Java, but:

> Uses structural reflection to transform at runtime
  — Simple model: Inline code before / after a bytecode
  — Inlined code is normal smalltalk code
  — Not much knowledge about bytecode needed
Example for Bytesurgeon I

> Goal: Logging Message send

```
example
  self test.
```

```
example
  Transcript show: 'sending #test'.
  self test.
```
Example for Bytesurgeon II

Goal: Log message send

with ByteSurgeon:

(Example>>#example) instrumentSend: [:send |
    send insertBefore:
        Transcript show: "sending #test".
]
Uses of ByteSurgeon at SCG

> Implementation of fast MethodWrapper
  — 35 lines of code

> Trace library for runtime tracing

> Back-In-Time Debugger

> Runtime analysis: test coverage
Problems of ByteSurgeon

> Performance
  - Faster then code / AST
  - But installation takes some time

> Abstractions too low level
  - Bytecode
  - We want to abstract away from bytecode and talk about instance variable access, message sending...
  - Not a good meta model
Geppetto

> Framework for behavioral reflection
> Build on top of ByteSurgeon
  — but abstracts from bytecode

> Fine grained scoping of reflection
  — spatial (where? and what?)
  — temporal (when?)

> Based on the Reflex Model (Eric Tanter)
Geppetto: Big Picture

AOP  Tracer  .......

Geppetto
ByteSurgeon
Squeak
Geppetto: Modell

Reflex
(Tanter OOPSLA 03)
Problem with Bytecode in Geppetto

> Bytecode is not a good meta model
> Lots of management infrastructure is needed
  — Hook composition
  — Synthesised elements (hooks) vs. original code
  — Mapping to source elements

> Bytecode is optimized
  — e.g. no ifTrue:
Beyond Text: A Meta Model for Methods

> We need a high-level meta model for methods
> This model needs to be causally connected
  — edit the model --> edit the system
> Text and Byte- (Binary-) code generated on demand
Beyond Text: A Meta Model for Methods

> Structure of method is implicit
  > Compile text (to AST)
  > Decompile bytecode (to IR or AST)

> Both text and bytecode are pretty low level

> Not suited for being the main representation
  > How to annotate text?
  > How to tag synthesised bytecode?

> Possible Model: AST
Many users

Method Meta Model

- Refactoring
- Aspects
- JIT
- Pluggable Typesystem
- Editor (pretty printer)
- Annotations
- Reflection
Explorations...

> Annotation framework
  - Nodes can be annotated
  - We can have any object as a (non-textual) annotation

> replace ByteSurgeon by AST based transformer
> Idea: Behavioral Reflection with Annotations
> Combine with AspectS for dynamic Aspects
Conclusion

- We have had a quick intro in Reflection
  - Squeak and how it enables reflection
- How to realize behavioral reflection
  - Bytesurgeon and Geppetto
  - Problems
- We need a Meta Model for Methods
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:

**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.