Efficient Multimethods in a Single Dispatch Language

Brian Foote

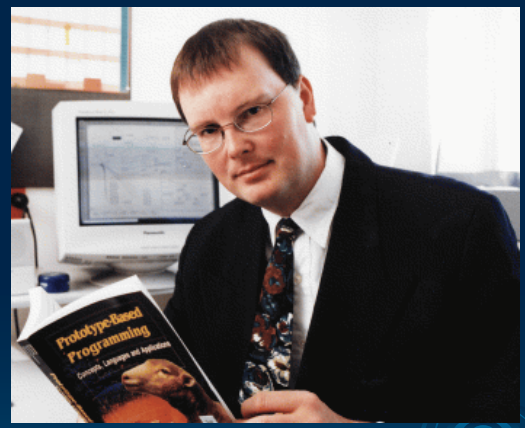
Ralph Johnson

Dept. of Computer Science

University of Illinois at Urbana-Champaign

201 N. Goodwin, Urbana, IL 61801, USA

foote@cs.uiuc.edu johnson@cs.uiuc.edu



James Noble

School of Mathematical and Computing Sciences

Victoria University of Wellington

P.O. Box 600, Wellington, New Zealand

kjx@mcs.vuw.ac.nz

Landmarks

- Motivation for Multimethods
- Syntax Matters
- A Tale of Two MOPs
- Two Implementations
- Enduring Significance

Let’s Get Serious

One day an Englishman, a Scotsman, and an Irishman walked into a pub together. They each called for a dram of whiskey. As they were about to enjoy their libations, three flies landed, one in each of their drinks, and became stuck inside the glasses. The Englishman pushed his shot away in disgust. The Scotsman fished the fly out of his glass, and continued drinking it, as if nothing had happened. The Irishman, too, picked the fly out of his drink, held it out over the glass, and started yelling, "SPIT IT OUT, SPIT IT OUT YOU BASTARD!!!!"

Motivation: A Simple Scotsman

```
public class Scotsman extends Carouser
{
   public void imbibe(ScotchWhiskey libation)
   {
       stomach.add(libation);
       bloodAlchoholLevel += 0.01;
       System.out.println("A wee dram...");
   }

   public void imbibe(IrishWhiskey libation)
   {
       stomach.add(libation);
       bloodAlchoholLevel += 0.00;
       System.out.println("Belfast Bog water...");
   }
```

A Simple Irishman

```
public class Irishman extends Carouser
{
   public void imbibe (ScotchWhiskey libation)
   {
        emptyStomach();
        System.out.
               println("Caledonian Swill...");
   }

   public void imbibe (IrishWhiskey libation)
   {
        stomach.add(libation);
        bloodAlchoholLevel += 0.01;
        System.out.println("Sure and begora...");
   }
```

A Simple, but Naïve Test

```
public void testOverloads()
{
      Irishman paddy = new Irishman();
      Scotsman angus = new Scotsman();
      System.out.
            println("--> testOverloads()...");
      paddy.imbibe(new IrishWhiskey());
      paddy.imbibe(new ScotchWhiskey());
      angus.imbibe(new IrishWhiskey());
      angus.imbibe(new ScotchWhiskey());
}
```

A Simple, Unsuccessful Variation

```
public void testBreakOverloads()
{
   Irishman paddy = new Irishman();
   Scotsman angus = new Scotsman();
   Carouser carouser = paddy;
   System.out.println("--> testBreakOverloads()...");
   Dram dram = new IrishWhiskey();
   // You can't really do this properly...
   carouser.imbibe(dram);
   carouser = angus;
   carouser.imbibe(new IrishWhiskey());
}
```

A Useless Overload

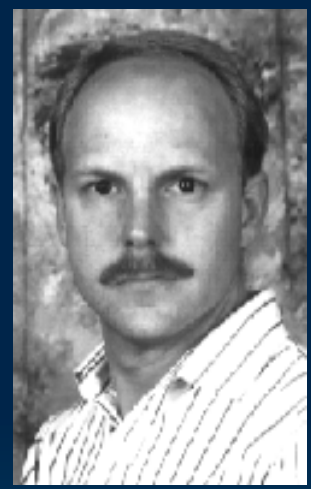
```
public void imbibe(Dram libation)
{
  System.out.
     println("Saints preserve us...");
}
```

A Type Case

```
public void imbibe(Dram libation)
{
    if (libation instanceof ScotchWhiskey)
    {
        emptyStomach();
        System.out.println("Caledonian Swill...");
    }
    else if (libation instanceof IrishWhiskey)
    {
        stomach.add(libation);
        bloodAlchoholLevel += 0.01;
        System.out.println("Sure and begora...");
    }
    else
        System.out.println("Mother of God...");
}
```

The Olfactory Method

Kent Beck: May be Best Remembered as the Man Who brought Scatology and Software Engineering together…



If it stinks, change it!

--Grandma Beck

Code Smells are (not so) subtle indications a piece of code is in need of attention… …and is a likely candidate for refactoring…

Double Dispatch

```
public class Irishman extends Carouser
{
    public void imbibe(Dram libation)
    {
        libation.whenImbibedByIrishman(this);
    }


public class IrishWhiskey extends Dram
{
   public void whenImbibedByIrishman(Irishman irishman)
    {
        irishman.stomach.add(this);
        irishman.bloodAlchoholLevel += 0.01;
        System.out.println("Sure and begora...");
    }
}
```

Dynamic Multidispatch?

```
public class Scotsman extends Carouser
{
   public void imbibe(<ScotchWhiskey> libation)
   {
       stomach.add(libation);
       bloodAlchoholLevel += 0.01;
       System.out.println("A wee dram...");
   }

   public void imbibe(<IrishWhiskey> libation)
   {
       stomach.add(libation);
       bloodAlchoholLevel += 0.00;
       System.out.println("Belfast Bog water...");
   }
```

Syntax Matters

CLOS:

```
(defmethod speak ((who animal))
   (format t "I'm an animal: ~A~%" who))
```

Dylan:

```
define method main (argv0 :: <byte-string>,
  #rest noise)
     puts("Hello, World.\n");
  end;
```

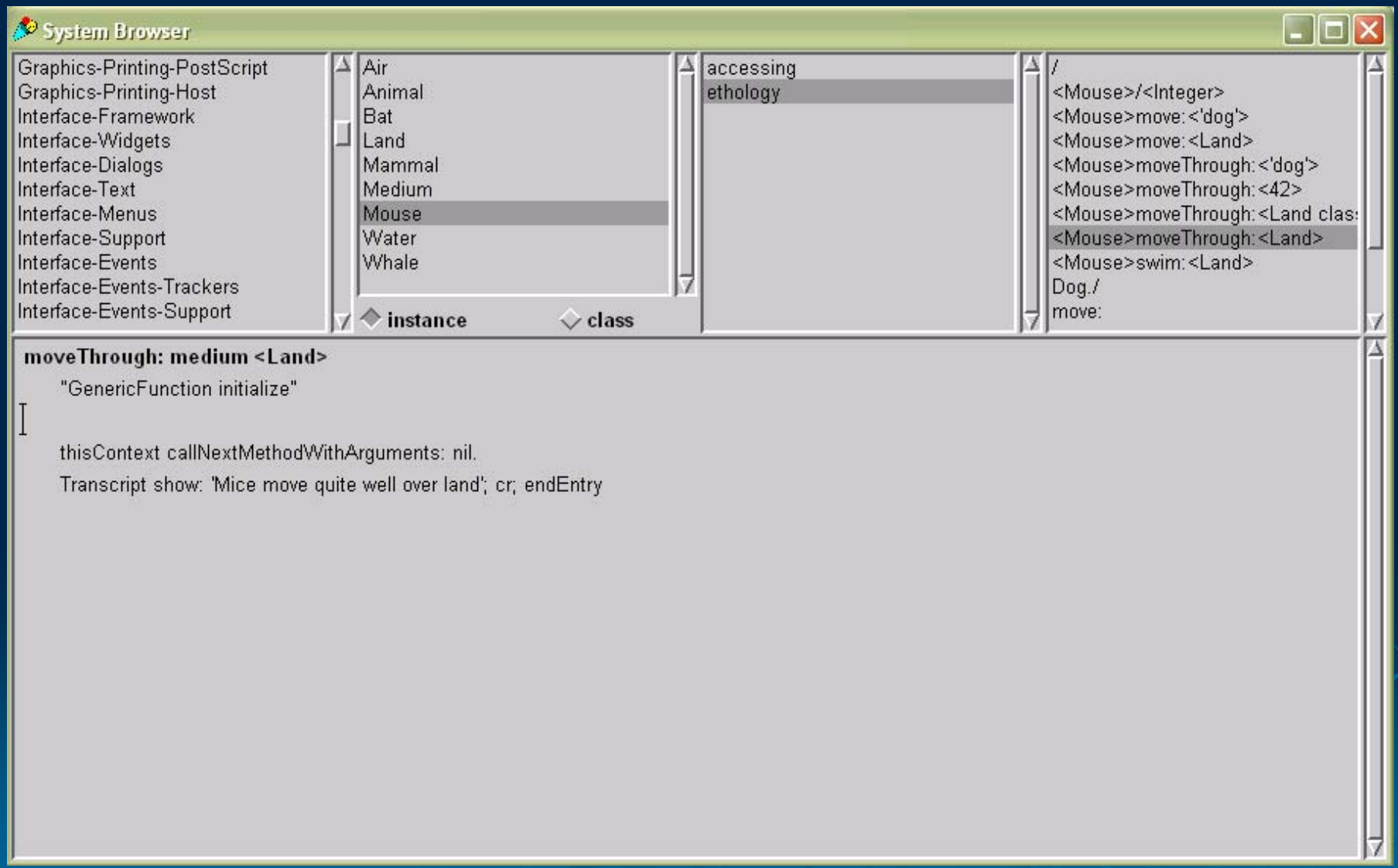
Cecil:

```
x@smallInt + y@smallInt
     { ^primAdd(x,y, {&errorCode | … })}
```

Multimethods in Smalltalk

```
ScreenDisplay>>draw: aGraphicalObject <Line>
    ”draw a line on a screen”
ScreenDisplay>>draw: aGraphicalObject <Arc>
    ”draw an arc on a screen”
```

Browsing Multimethods



Visitor: Before

```
ParseNode>>acceptVistor: aVisitor
  ^self subclassResponsibility
VariableNode>>acceptVistor: aVisitor
  ^aVisitor visitWithVariableNode: self
ConstantNode>>acceptVistor: aVisitor
  ^aVisitor visitWithConstantNode: self
OptimizingVisitor>>visitWithConstantNode:
  aNode
  ^aNode value optimized
OptimizingVisitor>>visitWithVariableNode:
  aNode
  ^aNode lookupIn: self symbolTable
```

Visitor: After

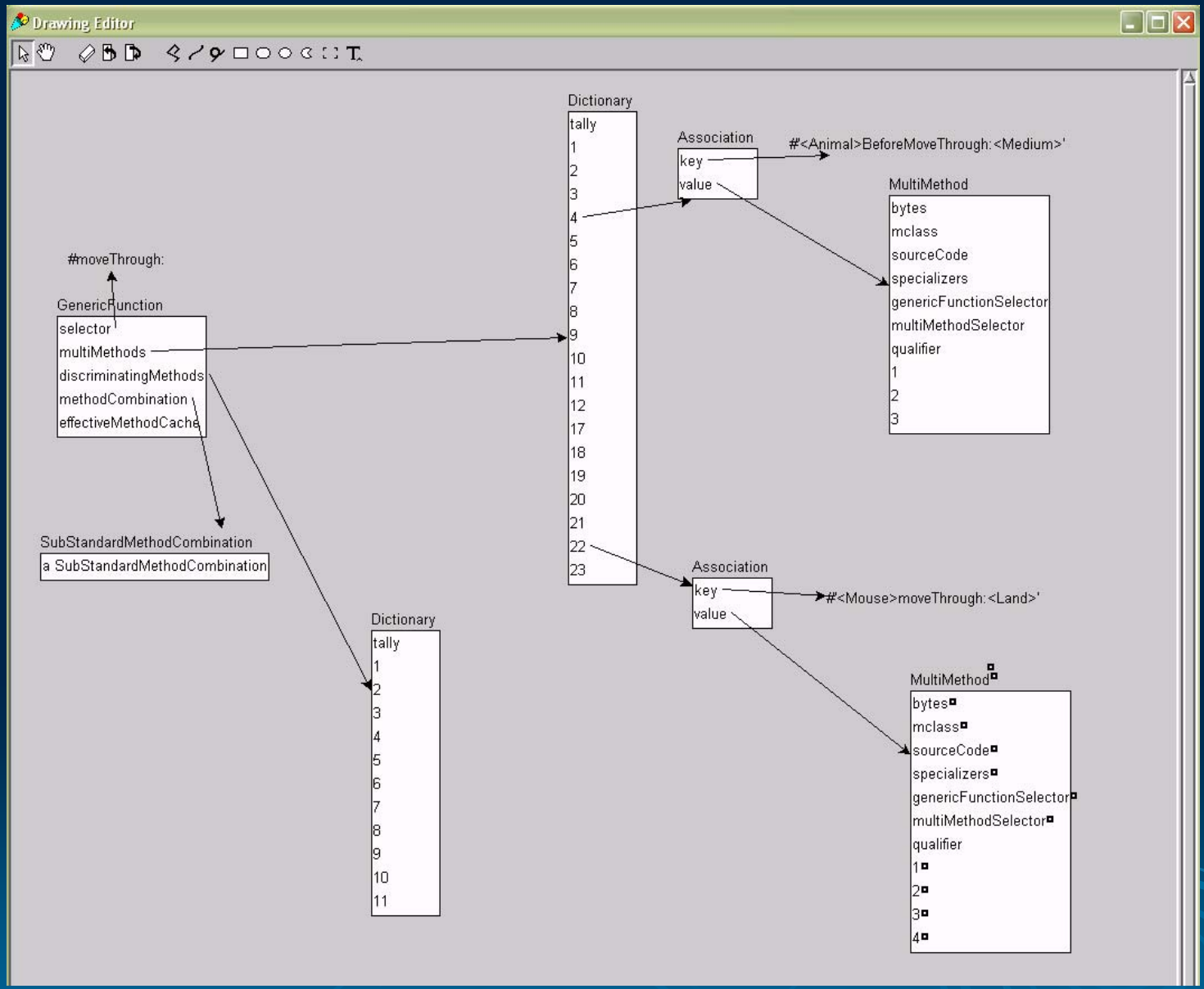
```
OptimizingVisitor>>visitWithNode: aNode
  <ConstantNode>
      ^self value optimized
OptimizingVisitor>>
      visitWithNode: aNode <VariableNode>
      ^aNode lookupIn: self symbolTable
```

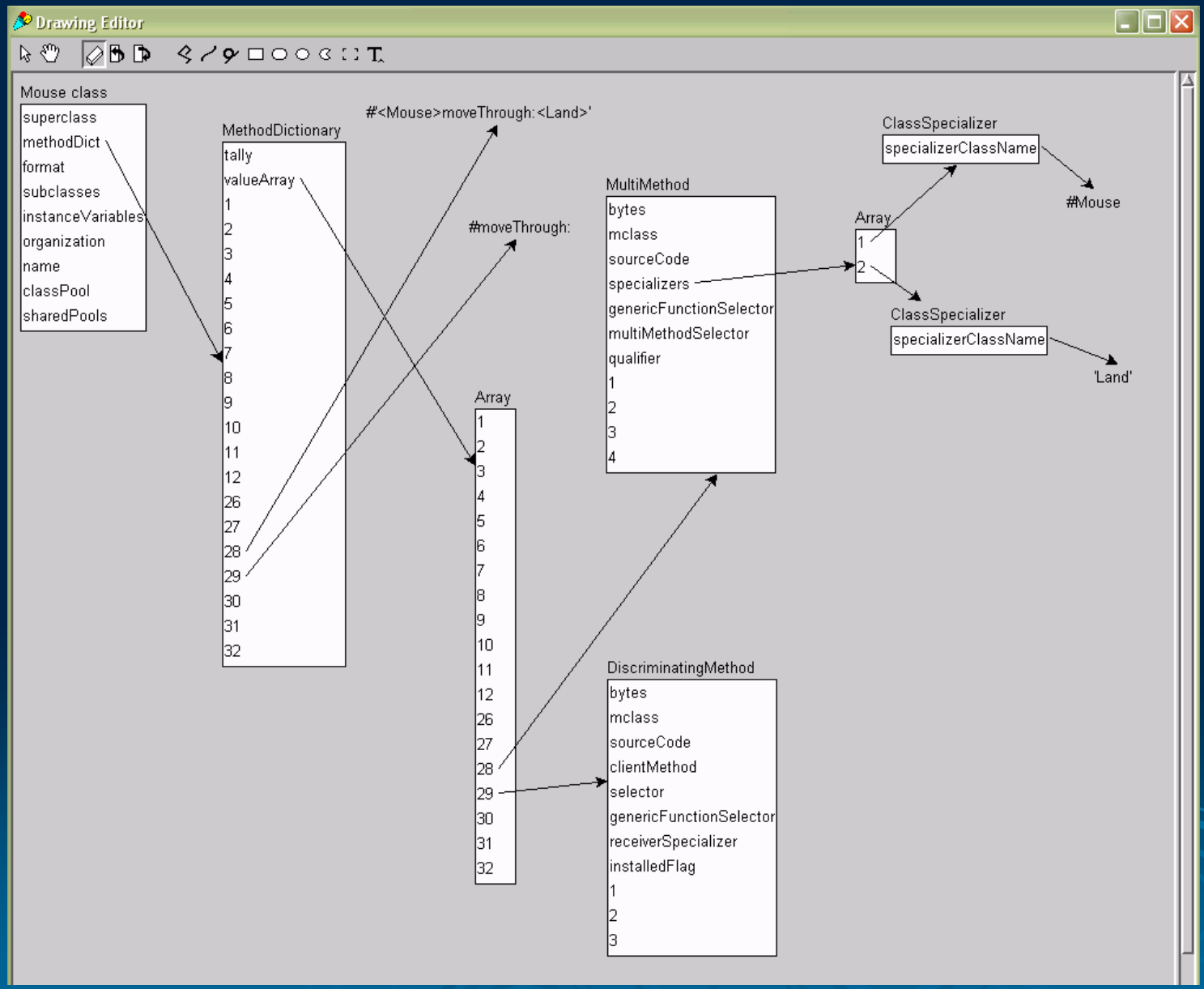
A Language Built of Objects

- Object
- Behavior
- ClassDescription
- Class
- Metaclass
- Method
- MethodDictionary
- CompiledMethod
- ByteArray
- Context
- MethodContext
- BlockContext
- Message
- Process
- ProcessScheduler
- Semaphore
- SharedQueue
- Compiler
- SystemDictionary

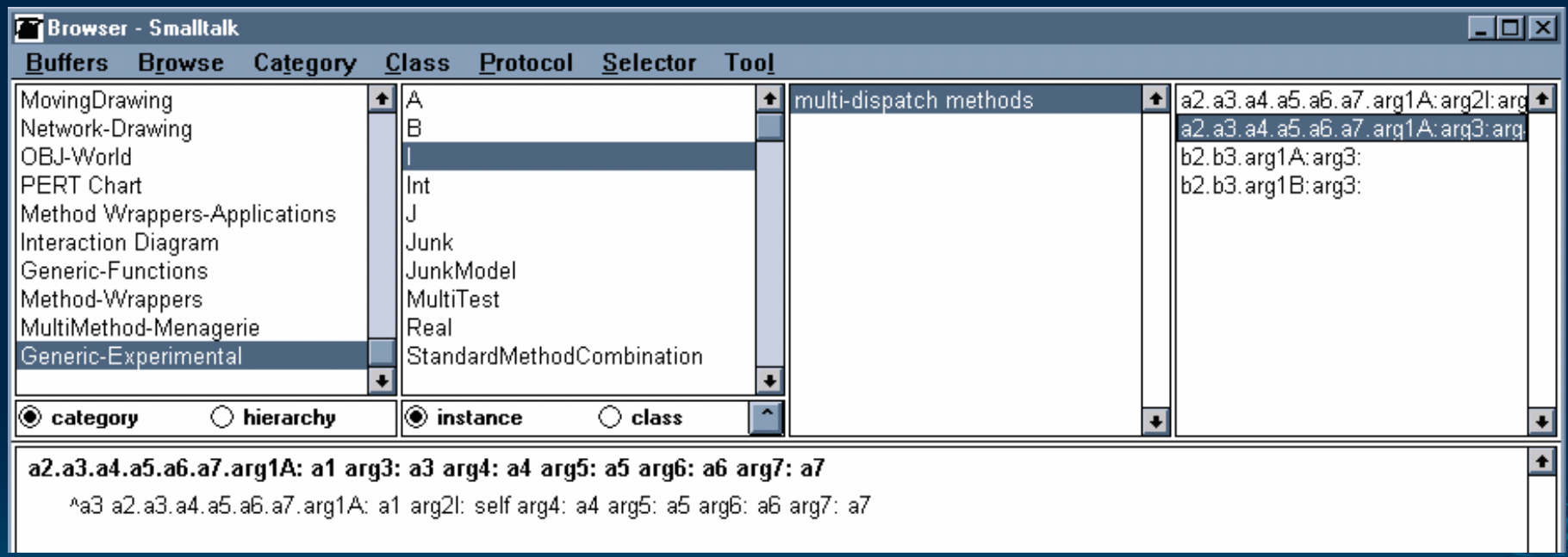
Objects We Built

- MultiMethod
- Specializer
- ClassSpecializer
- EqualSpeciealizer
- GenericMessage
- MethodCombination
- DiscriminatingMethod
- Qualifiers (#Before #After, etc.)
- SubStandardMethodCombination
- SimpleMethodCombination
- BetaMethodCombination
- DispatchingMethodCombination


Drawing Editor
Dictionary
tally
1
2
3
4
5
6
7
8
9
10
11
12
17
18
19
20
21
22
23
Association
key
value
#'<Animal>BeforeMoveThrough:<Medium>'
MultiMethod
bytes
mclass
sourceCode
specializers
genericFunctionSelector
multiMethodSelector
qualifier
1
2
3
#moveThrough:
GenericFunction
selector
multiMethods
discriminatingMethods
methodCombination
effectiveMethodCache
SubStandardMethodCombination
a SubStandardMethodCombination
Association
key
value
#'<Mouse>moveThrough:<Land>'
Dictionary
tally
1
2
3
4
5
6
7
8
9
10
11
MultiMethod
bytes
mclass
sourceCode
specializers
genericFunctionSelector
multiMethodSelector
qualifier
1
2
3
4


Drawing Editor
Mouse class
superclass
methodDict
format
subclasses
instanceVariables
organization
name
classPool
sharedPools
MethodDictionary
tally
valueArray
1
2
3
4
5
6
7
8
9
10
11
12
26
27
28
29
30
31
32
#'<Mouse>moveThrough:<Land>'
#moveThrough:
Array
1
2
3
4
5
6
7
8
9
10
11
12
26
27
28
29
30
31
32
MultiMethod
bytes
mclass
sourceCode
specializers
genericFunctionSelector
multiMethodSelector
qualifier
1
2
3
4
Array
1
2
ClassSpecializer
specializerClassName
#Mouse
ClassSpecializer
specializerClassName
'Land'
DiscriminatingMethod
bytes
mclass
sourceCode
clientMethod
selector
genericFunctionSelector
receiverSpecializer
installedFlag
1
2
3

N-Way Multidispatch



Generated Redispatching Methods

$$|D| = \sum_{i=1}^{n} \prod_{j=1}^{i} |Sj|$$

$$|D| = |S_1| + |S_2| \times |S_1| + |S_3| \times |S_2| \times |S_1| \ldots + |S_n| \times \ldots \times |S_2| \times |S_1| \times 2$$

Performance

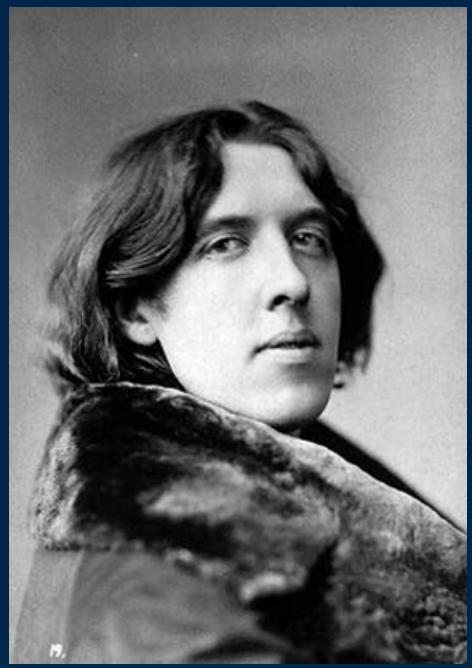
Dispatch Type	nanosec. min	nanosec. max	Ratio
1. Multidispatch (2 args)	521	524	1.00
2. Tare (^self) (1 arg)	90	120	0.20
3. Metaobjects (^self) (2 args)	597,000	624,000	1168
4. Metaobjects (super) (2 args)	679,000	750,000	1367
5. Metaobjects cached (2 args)	117,000	125,000	231
6. Dictionary (3 args)	13227	13335	25
7. Case (inline) (3 args)	10654	10764	20
8. Multidispatch (3 args)	633	779	1.35
9. Multidispatch (7 args)	1200	1221	2.32

Table I -- *Performance Results*
200MHz Pentium Pro
1,000,000 calls/multiple runs

Lessons

- The Beauty of Smalltalk
- The Elegance of the CLOS MOP
- Building Languages of Objects
- The Power of Multimethods

I Have Nothing to Declare



- End to End Argument
- Impact of Dynamic Types and Languages
- The Arrogance of Closed Worlds
- Reflection as a School of Architecture

Acknowledgements

- Ralph Johnson
- James Noble

- John Brant
- Don Roberts

- Richard P. Gabriel
- Andrew Black
- Stéphane Ducasse
- Christophe Dony
- Anonymous ECOOP Reviewers
- UIUC Software Architecture Group