Outline

- Introduction to Reflection
- Overview of MetaclassTalk
- Overview of MetaclassTalk Implementation
- Mixin Based Inheritance
- Metaclass Composition
- Conclusion

What is Reflection?

Structure of a Computational System

A Meta-system

A Reflective System
Definitions

- **Reflection**: Ability of a system to
  - observe itself = reason on itself
  - change itself = alter its own structure & behavior
- **Reify**: Represent some concept as an explicit entity
  - Building blocks of program and executor
- **A Reflective Language**
  - Language which constructs and "interpreter" are reified
  - Two programming levels

A System in OOPLs

- **Executor** = (virtual) machine, interpreter, …
  - language semantics
  - program loading & compilation
  - memory management
  - …
- **Program**
  - classes, methods, fields, messages …
- **Data**
  - objects: bank, clients, accounts, …

Degree of Reflection

A Reflective Language

- Language which constructs and "interpreter" are reified
- Two programming levels

OO Reflective Languages

- **Reification** = representing entities as objects
  - e.g. classes = instance of other classes: **Metaclasses**
- **base vs. meta-objects**
  - base-level ➔ meta-level
  - Base vs. Meta-Objects

Reflection is Useful

- **Development tools**
  - Browser, Debugger, Code Generators, …
- **Run-time Flexibility**
  - Quality of Service, Unplanned Evolution, …
- **Ease Software Development**
  - Generic Code ➔ Reuse
  - Separation of Concerns (Aspect-Oriented Programming)
- **Adapt and Extend the Language**
  - Asynchronous Communication,
  - Lazy memory allocation, …

Smalltalk Metaclasses are Implicit

What is MetaclassTalk?
**A Reflective extension of Smalltalk**

- **Explicit metaclasses**
  - New kinds of classes
  - Class Properties

- **Meta-objects (MOP)**
  - Objects structure management
  - Message dispatch
  - Per-object semantics

- **Goal**
  - Easing Experiments
  - Various Programming Paradigms

---

**MetaclassTalk “Conceptual” Kernel**

- **Object** inherits from **StandardClass**
- **MetaObject**
- **default semantic meta-object**

---

**MetaclassTalk Core MOP**

- Instance memory allocation (allocate) **class**
- Object creation (new = allocate and initialize) **semantic meta-object**
- Reading instance variables (at:of:) **semantic meta-object**
- Writing instance variables (at:of:put:) **semantic meta-object**

- Sending messages (send:...) **semantic meta-object**
- Receiving messages (receive:...) **meta-object**
- Method lookup (lookupFor:...) **class**
- Method evaluation (apply:...) **semantic meta-object**

---

**The Meta Link**

- **Links base-objects to semantic meta-objects**
  - Granularity = object
  - Sibling objects linked to different meta-objects

- An object can be linked to many semantic meta-objects
  - Semantic meta-objects should cooperate

- A semantic meta-object can be shared
  - i.e. linked to many base-objects

---

**Decomposition of a Message Dispatch**

1. Send #foo:bar: from: o1 to: o2 arguments: #(value1 value2)
2. Look up for #foo:bar: superfind false originClass: C2
3. Apply method FooBar to: o2 arguments: #(value1 value2)

---

**Example of Message Reception Control - 1**

- Person
- LogMetaObject
- **sayHello**

**Instance of**

- **default semantic meta-object**
- **log meta-object**

**console**

- **Hello Berne!!!**
**Example of Message Reception Control - 2**

- Log file
  - Message #sayHello received on 17th June at 5:30pm
- **console**
  - Hello Berne!!!

**Example of Message Sending Control**

- LogMetaObject
  - send: selector from: sender to: receiver arguments: ...
  - smo send: #nextPutAll from: joe to: aStream arguments: #(name) ...
  - printOn: aStream aStream nextPutAll: name

**Example of IV Read-Write Control**

- BankAccount
  - balance = newBalance
  - deposit: amount
    - newBalance := balance + amount
    - balance := newBalance

**Semantic Meta-objects Cooperation**

- 1 object is linked to 1 semantic metaObject
- Many semantic meta-objects for 1 object
  - meta-Objects cooperation
  - e.g. Chain of Responsibility Design Pattern

**Linguistic Change using Metaclasses - 1**

- SomeMetaClass subclass: #MyMetaClass
  - instanceVariablesNames: 'x y'
  - category: 'MetaclassTalk-Example'
  - metaclass: OtherMetaClass
- OtherMetaClass

**Linguistic Change using Metaclasses - 2**

- MyClass subclass: #MyClass
  - instanceVariablesNames: 'iv1 iv2'
  - category: 'MetaclassTalk-Example'
  - metaclass: MyClass
Example of a Linguistic Change

StandardClass
subclass of
SingletonClass
singleton
new
instance of
UndefinedObject
ProcessorScheduler
True
False

A Quick Tour Under the Hood
Overview of MetaclassTalk Implementation

Requirements
1. Full compatibility with Smalltalk
   - Portability: NO VM change
   - Use only Smalltalk reflective facilities
2. Hooks to support the MOP
   - IVs reads and writes
   - message sends
   - message receptions
3. Integration with Smalltalk
   - MetaclassTalk objects can interact with Smalltalk ones
   - MetaclassTalk classes can inherit from Smalltalk ones
   - Smalltalk classes can be "imported" into MetaclassTalk

Implementation
- Explicit Metaclasses
  - A new class builder
- Hooks for semantic reflection
  - Specific compilation process
  - Subclasses for Compiler, Parser
  - Abstract Syntax Tree Transformation
  - Method wrappers
- Stopping infinite regressions
  - Meta-circularity
  - Primitives

Hooks for IV reads and writes

Hooks for message sends - 1
4 Cases for Message Sends

- **Sender & Receiver = both ST objects**
  - No hooks

- **Sender & Receiver = both MT objects**
  - Hook at the sender method
  - Sender known at compile-time

- **Sender = ST object & Receiver = MT object**
  - Hook at receiver method
  - Sender known at run-time (thisContext)

- **Sender = MT object & Receiver = ST object**
  - Mimicking MetaclassTalk MOP

Hook for message reception

*Only for messages sent by Smalltalk objects*

```
bar: anObject

|sender|
sender := thisContext sender...

^self metaObject receive: #bar
  from: sender
to: self
arguments: #()
superSend: true
originClass: nil
```

Meta Link Infinite Regression

```
sayHello receive: #sayHello
receive: #receive:
...
base-level
```

Infinite Regressions

- **Meta Link**
  - Semantic meta-objects are objects,
    - controlled by some meta-meta-objects
    - and, meta-meta-objects are objects...

- **Instantiation Link**
  - Classes are objects,
    - are instances of some meta-classes
    - and, meta-classes are objects...

Stopping Meta Link Infinite Regression

```
primitive semantic meta-object
```
**Instance Link Infinite Regression**

- meta-meta-meta-level
- meta-meta-class
- meta-class
- instance of
- class
- base-level

---

**Stopping Instance Link Infinite Regression**

- Standard class
- instance of
- class

---

**A Word About Performance - 1**

- **Space overhead:**
  - method wrappers

- **Time overhead:**
  - jumps to the meta-level = Interpretation
  - message sending
    - $\sim 0.04\%$ of Smalltalk message sending
  - IV access
    - $\sim 15.8\%$ of Smalltalk IV access

---

**A Word About Performance - 2**

- **Selective hook introduction**
  - Limit overhead to where reflection is needed

- **By default:**
  - No hooks on "special messages"
    - i.e. macros, (e.g. ifTrue:, whileFalse:, …)
  - No hooks on meta-level code
    - e.g. StandardClass and its subclasses

---

**Mixin Based Inheritance In MetaclassTalk**

---

**Why Mixin-Based Inheritance?**

- **Context:**
  - Single inheritance
  - Unrelated hierarchies
  - Same Properties

- **Problem:**
  - Reuse shared properties
  - Avoid code duplication
Example of Code Duplication

PositionableStream

ReadStream

next

next: anInteger

| ans endPosition |

endPosition := …

next

WriteStream

next

next shouldNotImplement

next

next: anInteger

| ans endPosition |

endPosition := …

Mix-in--Based Inheritance

Mixin

B

C

D
equals from mixin

mixins

Object

A

E

F

G

inherits from mixin

X

iv1

iv2

foo

bar

Mixin = Subclass Generator

[Bracha & Cook 90]

3 metaclasses (i.e. 3 class properties):

- Mixins
- Classes generated by mixins
- Composite classes "inherit" from mixins

Generated classes are updated on mixin change

Use of Smalltalk reflective facilities

- Method compilation
- Class building
- Adds & removals of methods and instance variables

Example of MetaclassTalk Mixins

ColoredBoundedPoint

Point subclass: #ColoredBoundedPoint

instanceVariableNames: ' ' category: 'Mixin-Example' metaclass: CompositeClass.

instanceVariableNames:

poolDictionaries:

category:

Colored

Bounded

CompositeClass

Point

mixins: {Colored. Bounded}

mixin subclass of

mixins

generated by mixins

Mixins Implementation in MetaclassTalk

X

iv1

iv2

foo

bar

Mixin = Subclass Generator

[Bracha & Cook 90]

Generated Class Hierarchy

Point

GColored

mixins: {Colored. Bounded}

GBounded

mixins: {Colored. Bounded}

ColoredBoundedPoint

mixins: {Colored. Bounded}
Refactoring the Stream Hierarchy

ReadWriteStream

ReadStreamMixin

next: anInteger

endPosition := ...

Need for Metaclass Composition

Why?
- Reuse
- A class can have many properties

How?
- Mixins at the metaclass level

Problems to deal with?
- Compatibility

Example: Mixin-Metaclases for Booleans

AbstractClass
    new
    inherits: from mixin

BooleanClass
    instance of

TrueClass

FalseClass

Example of inter-level communication

Object>>printOn: aStream
    |title|
    title := self class name.
    aStream nextPutAll: ...

Collection class>>with: anObject
    |newCollection|
    newCollection add: anObject...
The Upward Compatibility issue

foo is NOT understood by B

The Downward Compatibility issue

bar is NOT understood by aZ

Undesirable Class Properties in Smalltalk

Metaclass level

Class level

Instance level

Compatibility & Class Specific Properties

Metaclass level

Class level

Compatibility, Class Properties & Mixins

Metaclass level

Class level

Example: Refactoring the Boolean Hierarchy
**Conclusion**

- Reflective Programming Languages
  - Linguistic reflection
  - Semantic reflection

- MetaclassTalk extends Smalltalk
  - Unleash Metaclasses (Linguistic Reflection)
  - Semantic Reflection

- Metaclasses are useful
  - Class properties – new "kinds" of classes
  - e.g. Mixin based inheritance (3 class properties)

**Summary - 1**

**Summary - 2**

- Metaclass Composition using Mixins
  - Class-Metaclass compatibility
  - No undesirable class properties propagation

- Other ongoing experiments with MetaclassTalks
  - Aspect-Oriented Programming,
  - Software Components,
  - Distribution (Web Services, …),
  - Multi-Agent Systems,
  - Mobile code…

- An implementation is available for Squeak

**Some Future Works**

- Extend mixin-based inheritance
  - Traits approach for methods composition
  - Instance variables composition

- OO Programming without "traditional" inheritance
  - Mixin based inheritance only!

- Refactoring Smalltalk libraries
  - Explicit metaclasses + Mixins
  - New kernel

**Thanks for your attention**

**Questions? Comments?**

Documents & Download
http://csl.ens-mdouai.fr/MetaclasseTalk