AOP using Reflection in Smalltalk
The MetaclassTalk Experiment

Noury Bouraqadi
Computer Science Laboratory (CSL)
Ecole des Mines de Douai
France

Outline
- What is AOP?
- What is Reflection?
- What is MetaclassTalk?
- AOP Using MetaclassTalk
- Conclusion

What is AOP?

From OOP to AOP
- AOP is a new paradigm for building programs
  - Led to AOSD
- AOP does not discard OOP
  - but, AOP is not bound to OOP
- AOP addresses 2 OOP limitations
  - Code tangling
  - Cross-cutting

An E-Business Application

Code Tangling
RemoteObject subclass: #Book
instanceVariableNames: 'title author price lock'…
initialize
lock := Semaphore forMutualExclusion.
self price: ('myDatabase priceFor: self').
super initialize.

"self start port listener = remote communication + marshaling"
"register into a name registry"

price
|currentPrice|
lock critical: [currentPrice := price).
Transcript cr; show: self printString.

"myDatabase setPrice: newPrice for: self"
Transcript cr; show: 'DataBase price: ', ('myDataBae priceFor: self') printString.
AOP Concepts

- **Base Code**
  - Most functionalities
- **Aspect**
  - Unit of code that describes a single "global" property
  - Examples: Remote communication, Synchronization, ...
- **Join points**
  - Points of the base code execution flow
  - Example: sending message m to object o
- **Weaving**
  - Assembling aspects with base code at join points

What is Reflection?

Structure of a Computational System

A Meta-system

Cross-cutting

AOP

Persistence

Distribution

Synchronisation

Order

Credit-Card

Bank

Shop

Client

Debug

Program

System

Data

Program

Executor

Part of the World

Part of the World

Software & hardware

Interpreter, (virtual) machine, peripheral…

Reasons & acts upon

Reasons & acts upon
**A Reflective System**

- **System**
  - Data
  - Program
  - Executor

  self "aware"

  Part of the World

  reasons & acts upon

**Definitions**

- **Reflection**: Ability of a System to
  - observe itself
  - reason on itself
  - change itself
    - Alter its structure (program)
    - Alter its behavior (executor)

- **Reify**: Represent some concept as an explicit entity
  - Building blocks of program and executor

- **A Reflective Language**
  - Language which constructs and "interpreter" are reified

**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**

- **Reification**: representing entities as objects
  - e.g. classes
  - is an object
    - controls one or more base-objects
      - i.e. responsible of the execution of messages, field accesses, ...

**Example of Meta-Object Usage - 1**

- Person
  - sayHello

- LogMetaObject
  - receive aMessage
    - "write a log in some file"
    - "execute the right method"

**Example of Meta-Object Usage - 2**

- Person
  - sayHello

- LogMetaObject
  - receive aMessage
    - "write a log in some file"
    - "execute the right method"

- Log file
  - Message sayHello received on 6th June at 2pm

  Hello VUB!!!
The Meta Link

- Meta-link = link objects to their meta-objects
  - Granularity = object
  - Siblinging objects linked to different meta-objects
- An object can be linked to many meta-objects
  - Meta-objects should cooperate
- A meta-object can be shared
  - i.e. linked to many base-objects

Reflective Towers & Infinite Regression

- Meta-meta-objects
  - Meta-meta-objects are objects
    - controlled by meta-meta-meta-objects
    - Infinite tour
- Stopping the infinite regression
  - Primitive/Default meta-object

Reflection is Useful

- Development tools
  - Browser, Debugger, Code Generators, …
- Run-time Flexibility
  - Quality of Service, Unplanned Evolution, …
- Adapt and Extend the Language
  - Mixin based inheritance, Asynchronous Communication, Lazy memory allocation…
- Ease Software Development
  - Generic Code ⇒ Reuse
  - Separation of Concerns

What is MetaclassTalk?

A Reflective extension of Smalltalk

- Explicit metaclasses
  - New kinds of classes
  - Class properties
- Meta-object (MOP)
  - Objects structure management
  - Message dispatch
- Goal
  - Easing Experiments
  - Various Programming Paradigms

MetaclassTalk "Conceptual" Kernel
**MetaclassTalk MOP**

- Instance memory allocation (allocate)
- Object creation (new = allocate or initialize)
- Reading instance variables (atIV:at:) or (atIV:of:put:)
- Writing instance variables (atIV:of:put:)
- Sending messages (send:...)
- Receiving messages (receive:...)
- Method lookup (lookupFor:...)
- Method evaluation (apply:...)

**Decomposition of a Message Dispatch**

1. **Instance memory allocation**
   - **Allocate**: new = allocate
   - **Initialize**

2. **Object creation**
   - **New = Allocate**
   - **Initialize**

3. **Reading instance variables**
   - **AtIV:AtIV:Of:Of:**

4. **Writing instance variables**
   - **AtIV:Of:Put:**

5. **Sending messages**
   - **Send:**

6. **Receiving messages**
   - **Receive:**

7. **Method lookup**
   - **LookupFor:**

8. **Method evaluation**
   - **Apply:**

**Example of Message Sending Control**

```plaintext
joe printOn: aStream
aStream nextPutAll: name
```

```plaintext
send: selector from: sender to: receiver arguments: arg...
Transcript cr; show: 'Sending msg ', selector.
^ super send: selector from: sender to: receiver arguments: arg...
```

**Example of IV Read-Write Control**

```plaintext
account atIV: ivIndex of: anObject put: value
self halt.
^ super atIV: ivIndex of: anObject put: value
```

```plaintext
index of IV balance
```

**Meta-Objects Cooperation**

- 1 object is linked to 1 metaObject
- Controlling 1 object by many meta-objects
  - Meta-Objects cooperation
  - E.g. Chain of Responsibility Design Pattern

**AOP using MetaclassTalk**
A First Natural Separation

How to Isolate & Weave Aspects?

Issues & Solutions

An E-Business Application

MetaObjects for a Synchronization Aspect

Configuration Script for Synchronization Aspect
### MetaObjects for a Persistence Aspect

**PersistMetaObject**

- `atIV: ivIndex of: anObject put: value` super `atIV: ivIndex of: anObject`
- "storage updateObject: anObject"

### MetaObjects for a Debug Aspect

**BreakPointMetaObject**

- `atIV: ivIndex of: anObject` ifTrue: [self halt].
- "super atIV: ivIndex of: anObject"

### Configuration Script for Persistance Aspect

- **Link every book and order to a new persistance meta-object**
  - i.e. when a new instance (book or order) is created
    1. `PersistMeta`
    2. `persistMeta := PersistMetaObject new.`
    3. `newInstance addMetaObject: persistMeta`

### Configuration Script for a Debug Aspect

- **Link orderX to a log meta-object**
  - `orderX addMetaObject: LogMetaObject new`
- **Link all books to a single break point meta-object**
  - `Book addInstanceMetaObject: b2`.

### Our E-Business Application in MetaclassTalk

#### Base Code

- **Base Objects Meta-Objects**
- **Synchro. Aspect**
- **Persist. Aspect**
- **Debug Aspect**

#### Conclusion
Summary

- Reflection Does Support AOP
  - NO new language construct
  - Reusable aspects: generic meta-objects
  - Flexibility: dynamic meta-objects/aspects change
  - Aspects conflicts = meta-object composition problem

- MetaclassTalk eases experiments
  - AOP (using Meta-Objects)
  - Mixins (using Metaclasses)
  - …
  - An implementation is available for Squeak 3.2

Some Future Works

- Other experiments
  - Distribution
  - Software Components
  - Multi-Agents Systems
  - (Strong) Mobility?…

- Enhancing MetaclassTalk
  - Improving performance (Code inlining?, Specific VM?)
  - Refactoring the full Smalltalk library
  - Migration to the latest Squeak release