

1

# MetaclassTalk & Metaclass Composition

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

2

## Outline

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

3

## What is Reflection?

4

## Structure of a Computational System

5

## A Meta-system

6

## 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
    - executor → meta-level
    - program → base-level

### A System in OOPs

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

more or less reified  
 ↓  
 Degree of Reflection

### OO Reflective Languages

- **Reification = representing entities as objects**
  - e.g. classes = instance of other classes: **Metaclasses**
- **base vs. meta-objects**
  - meta-level → meta-objects
  - base-level → base-objects
- **Two kinds of meta-objects:**
  - **Linguistic meta-objects** = Language constructs
    - e.g. classes, methods, fields, name spaces...
  - **Semantic meta-objects** = Language semantics
    - e.g. message dispatch, memory allocation, field access...

### Reflection is Useful

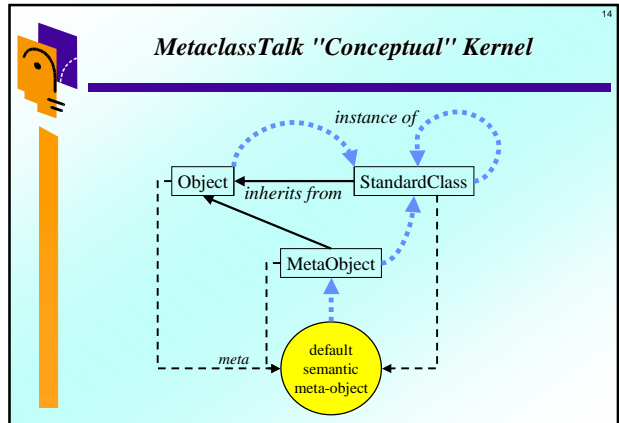
- **Development tools**
  - Browser, Debugger, Code Generators, ...
- **Run-time Flexibility**
  - Quality of Service, Unplanned Evolution, ...
- **Ease Software Development**
  - Generic Code ⇔ Reuse
  - Separation of Concerns (Aspect-Oriented Programming)
- **Adapt and Extend the Language**
  - Asynchronous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

### What is MetaclassTalk?

### Smalltalk Metaclasses are Implicit

### 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



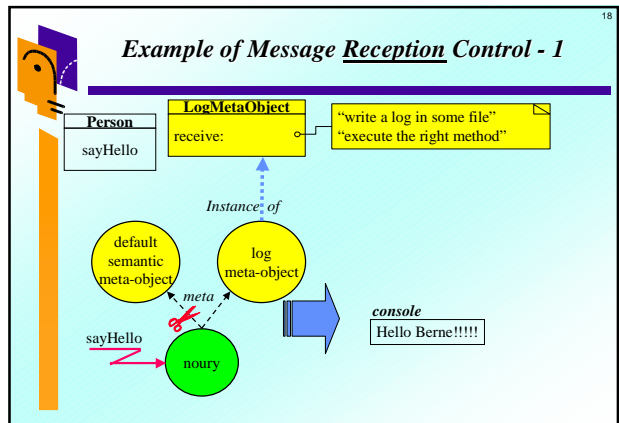
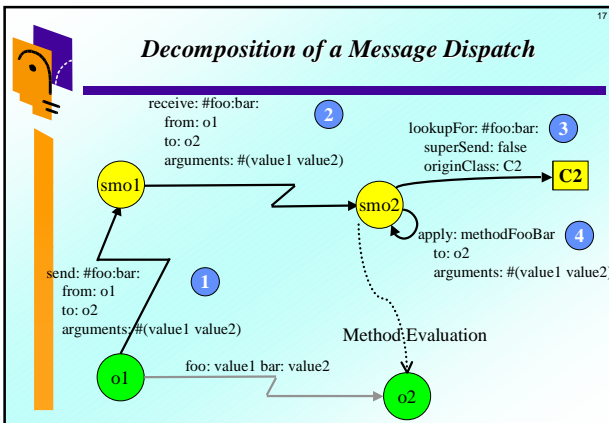
### MetaclassTalk Core MOP

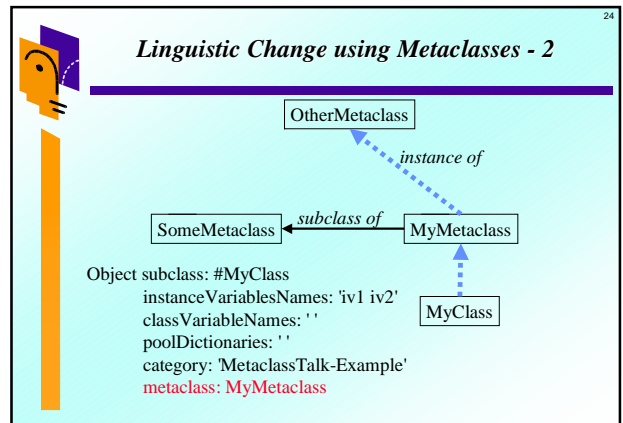
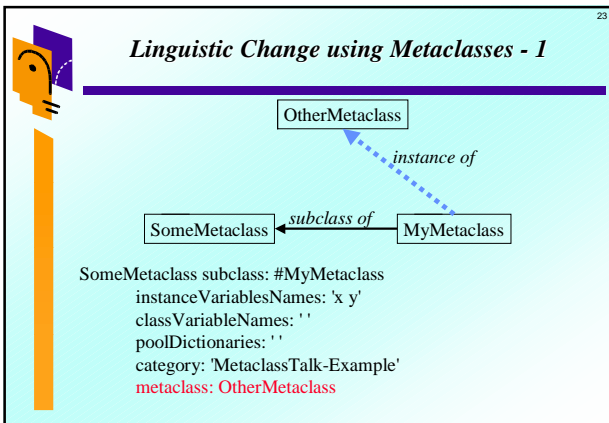
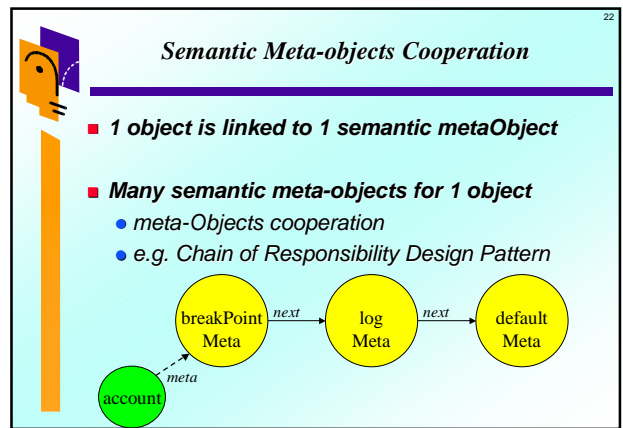
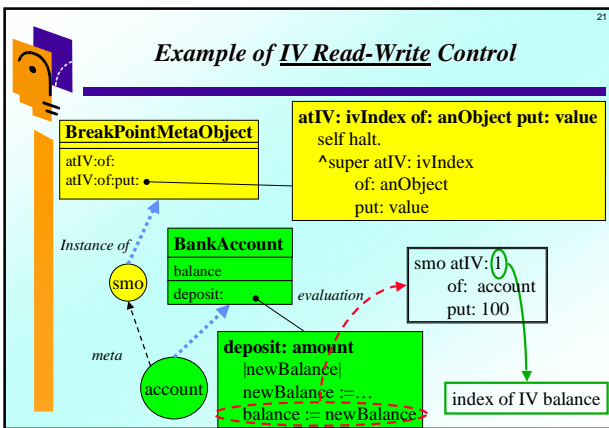
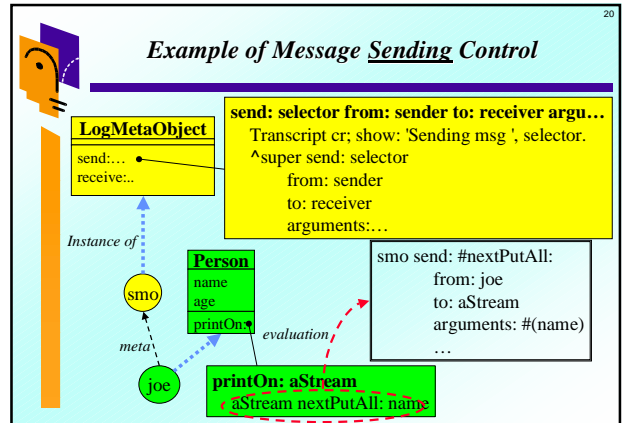
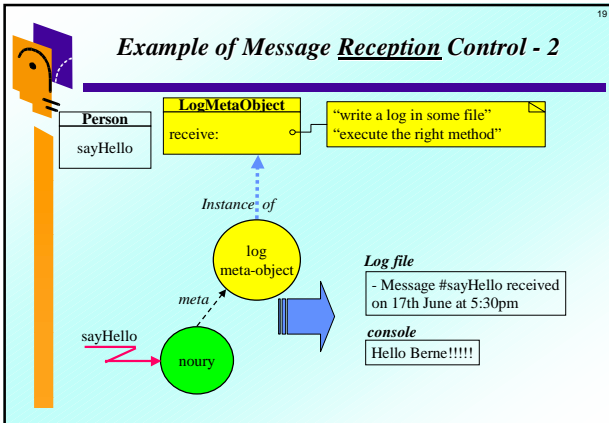
■ instance memory allocation (allocate)	class
■ object creation (new = allocate o initialize)	class
■ reading instance variables (atV:of:)	semantic meta-object
■ writing instance variables (atV:of:put:)	semantic meta-object

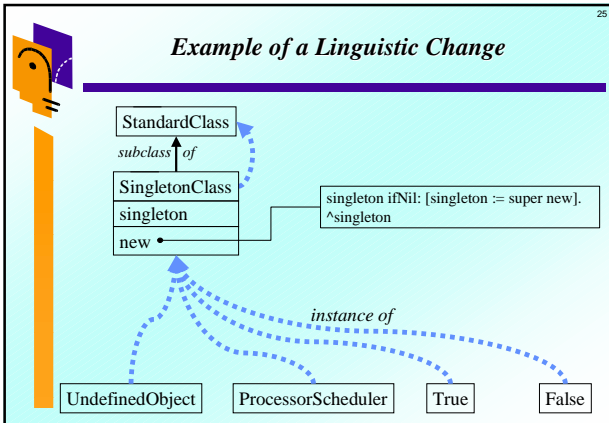
  

■ sending messages (send:...)	semantic meta-object
■ receiving messages (receive:...)	semantic meta-object
■ method lookup (lookupFor:...)	class
■ method evaluation (apply:...)	semantic meta-object

- ### 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





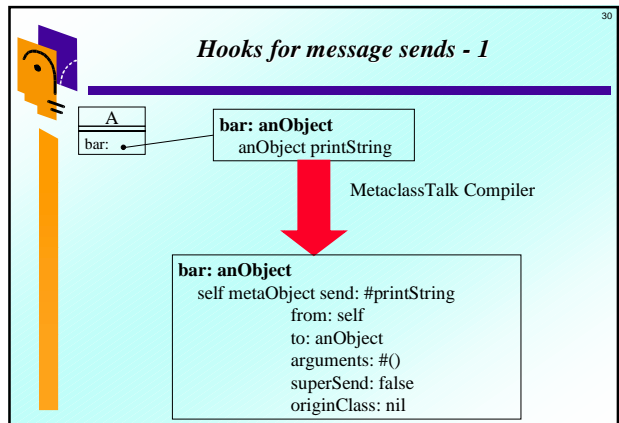
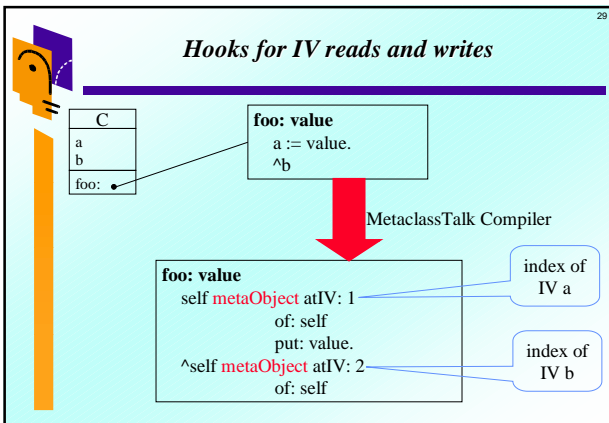


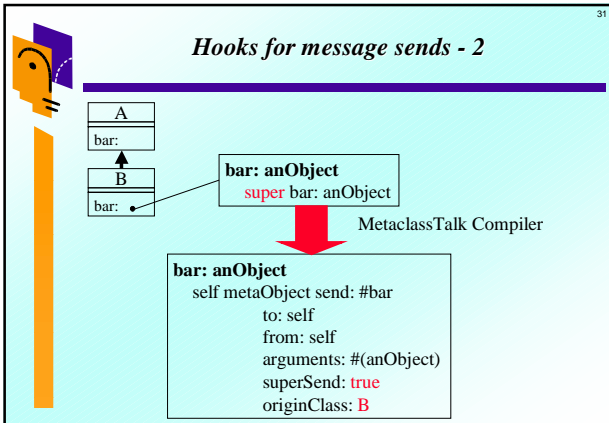
## A Quick Tour Under the Hood

### Overview of MetaclassTalk Implementation

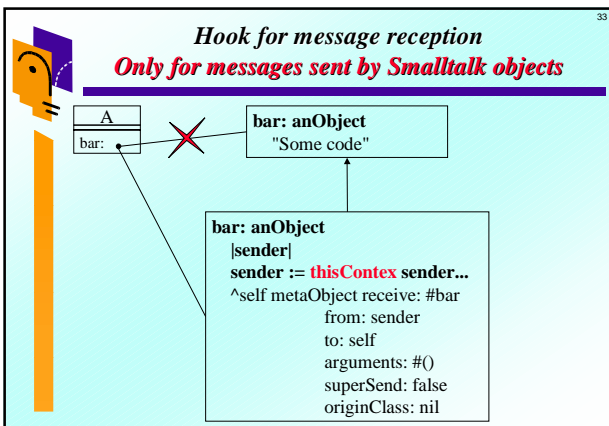
- ### Requirements
- Full compatibility with Smalltalk**
    - Portability: **NO** VM change
    - Use only Smalltalk reflective facilities
  - Hooks to support the MOP**
    - IVs reads and writes
    - message sends
    - message receptions
  - Integration with Smalltalk**
    - MetaclassTalk objects 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

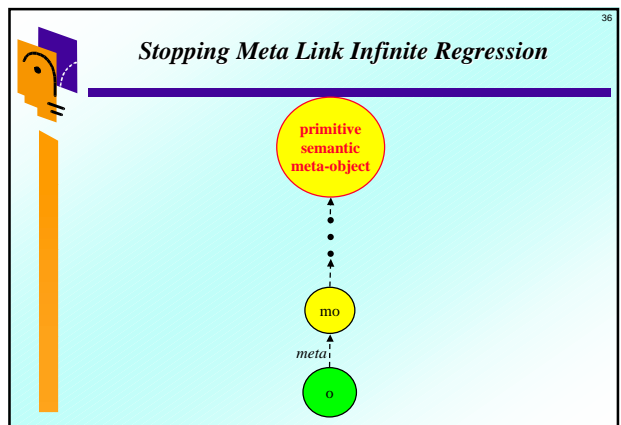
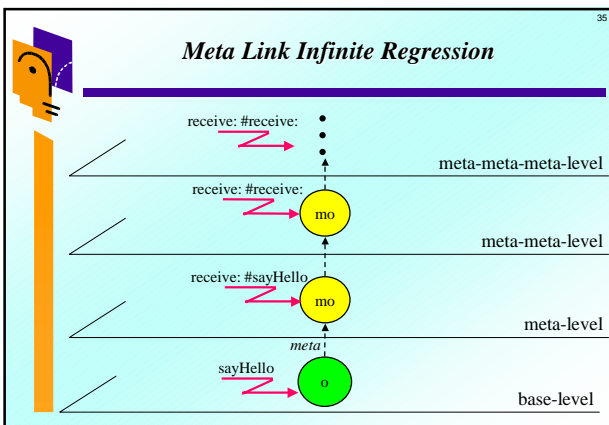


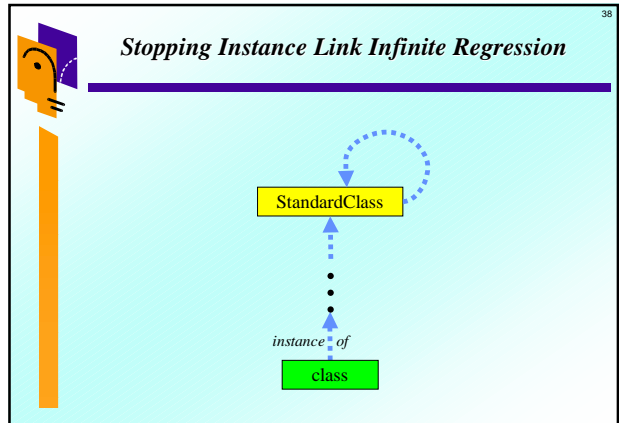
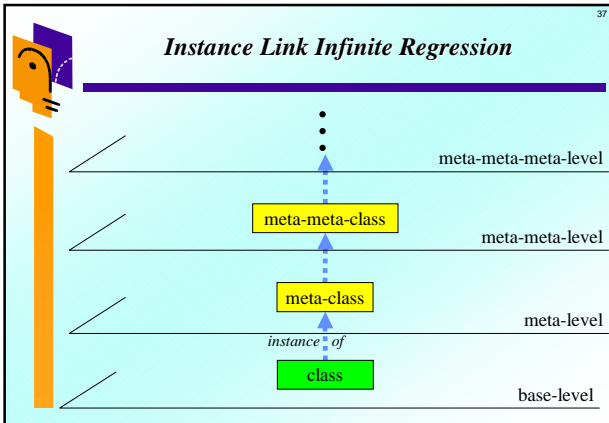


- ### 4 Cases for Message Sends
- **Sender & Receiver = both ST objects**
    - No hooks
  - **Sender & Receiver = both MT objects**
    - Hook at the sender method
    - Sender known at compile-time
  - **Sender = ST object & Receiver = MT object**
    - Hook at receiver method
    - Sender known at run-time (*thisContext*)
  - **Sender = MT object & Receiver = ST object**
    - Mimicking MetaClassTalk MOP



- ### 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...

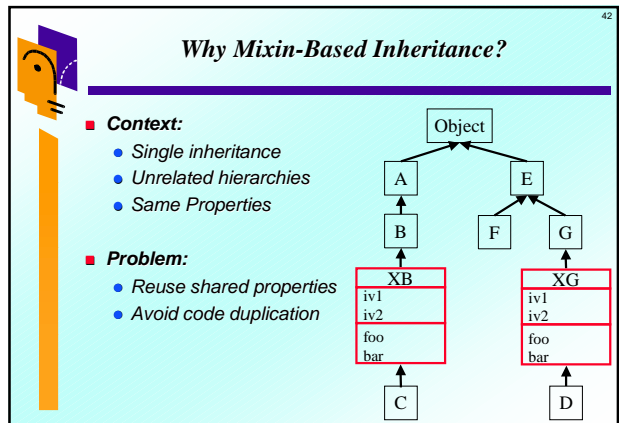


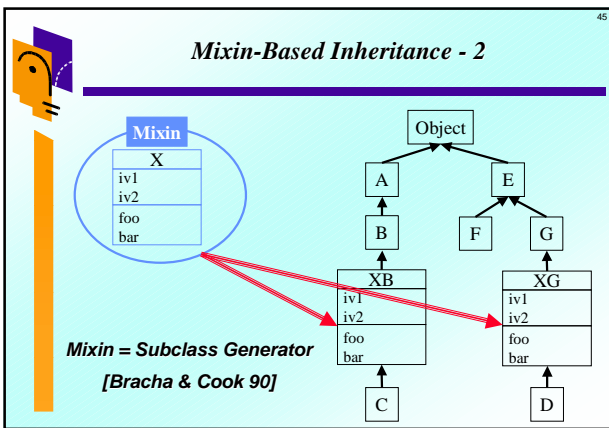
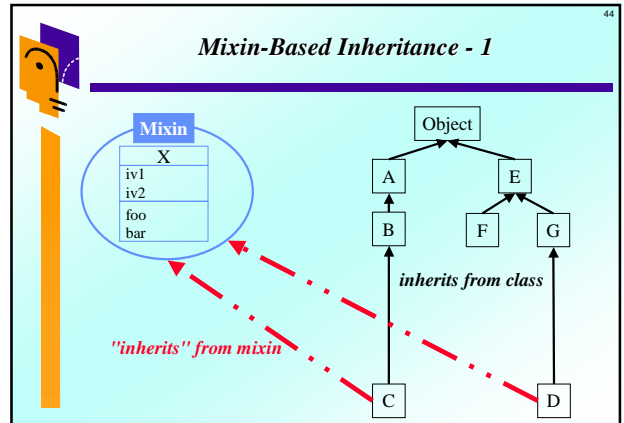
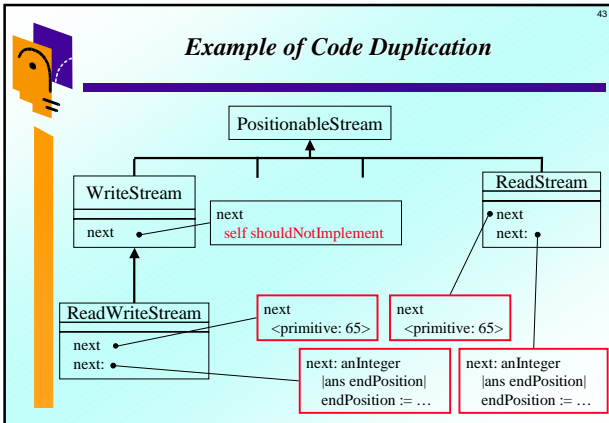


- ### A Word About Performance - 1
- **Space overhead:**
    - method wrappers
  - **Time overhead :**
    - jumps to the meta-level = Interpretation
    - message sending
      - ~0.04% of Smalltalk message sending
    - IV access
      - ~15.8% of Smalltalk IV access

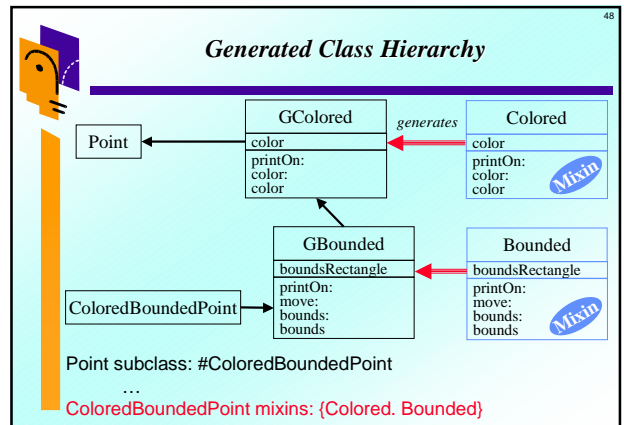
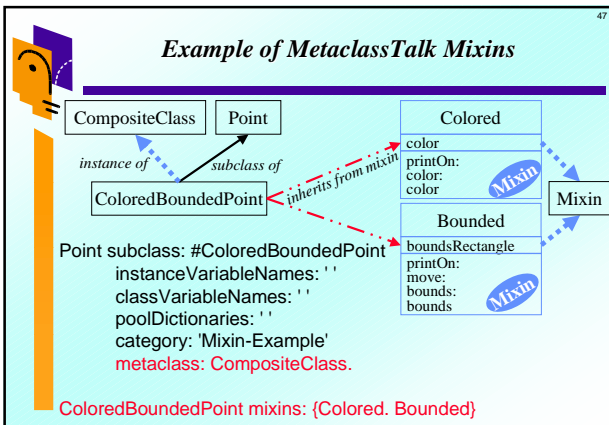
- ### A Word About Performance - 2
- **Selective hook introduction**
    - Limit overhead to where reflection is needed
  - **By default:**
    - No hooks on "special messages"
      - i.e. macros, (e.g. ifTrue:, whileFalse:, ...)
    - No hooks on meta-level code
      - e.g. StandardClass and its subclasses

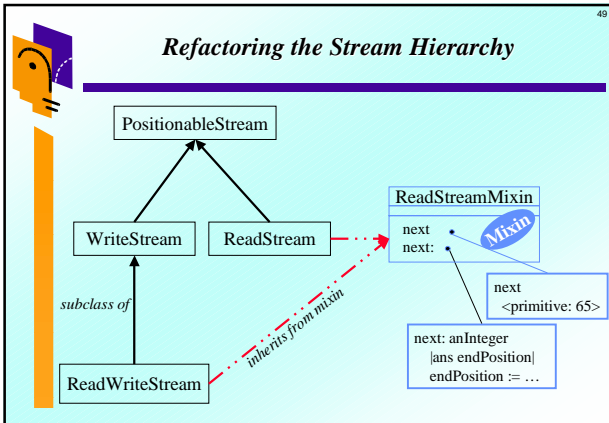
### Mixin Based Inheritance In MetaclassTalk



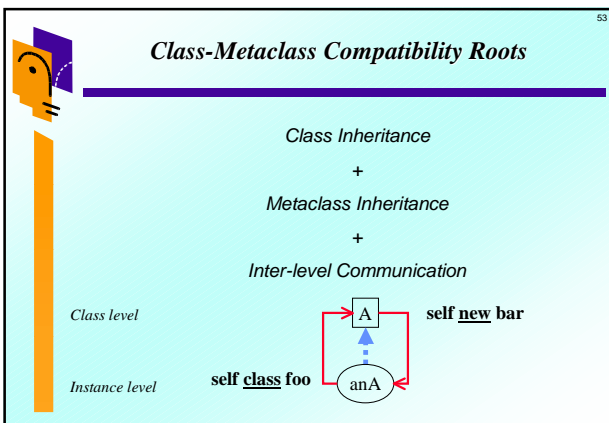
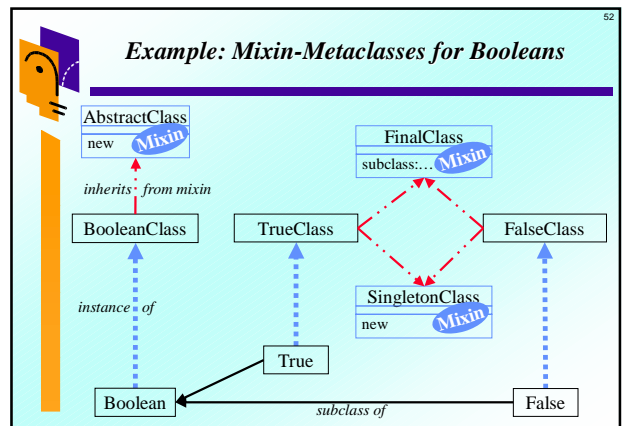


- ### Mixins Implementation in MetaclassTalk
- 3 metaclasses (i.e. 3 class properties):
    - Mixins
    - Classes generated by mixins
    - Composite classes "inherit" from mixins
  - Generated classes are updated on mixin change
  - Use of Smalltalk reflective facilities
    - Method compilation
    - Class building
    - Adds & removals of methods and instance variables

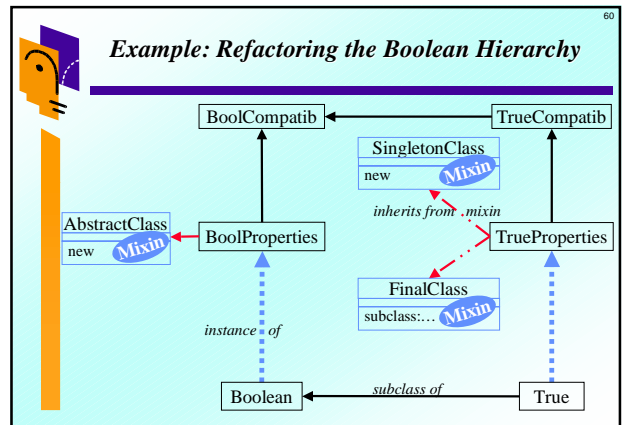
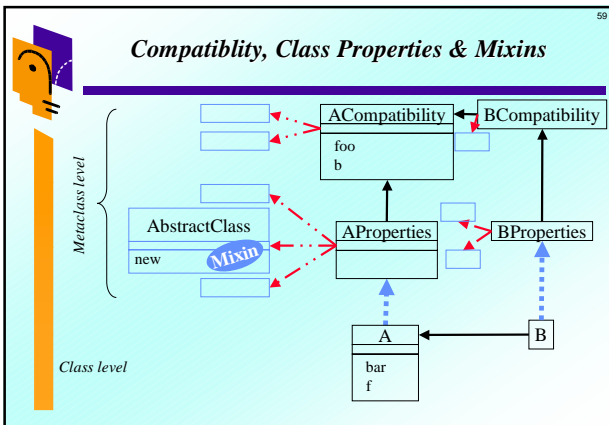
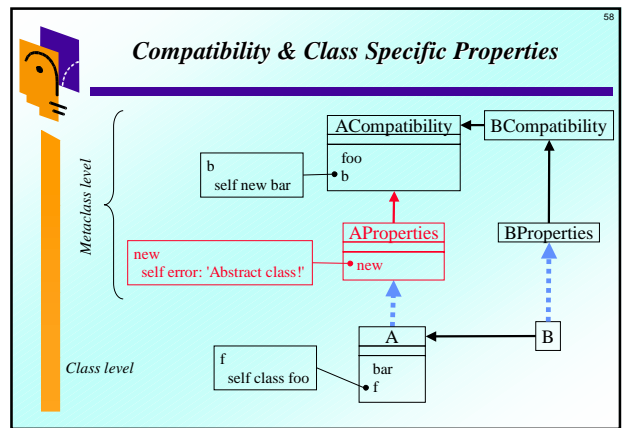
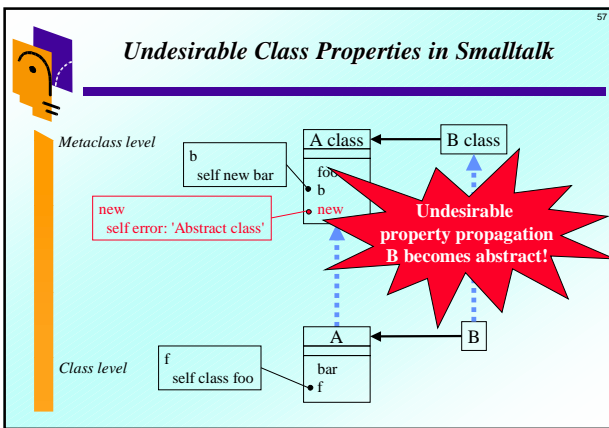
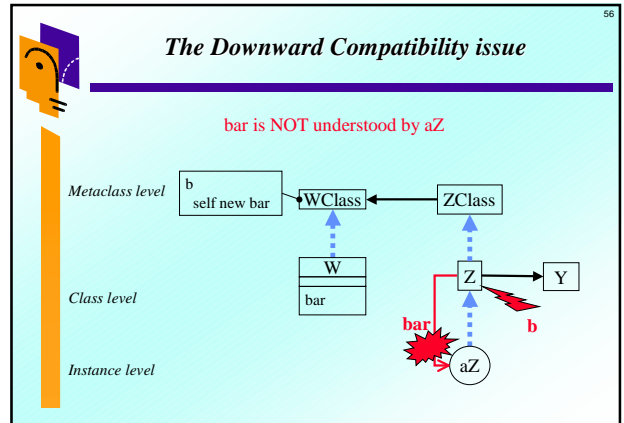
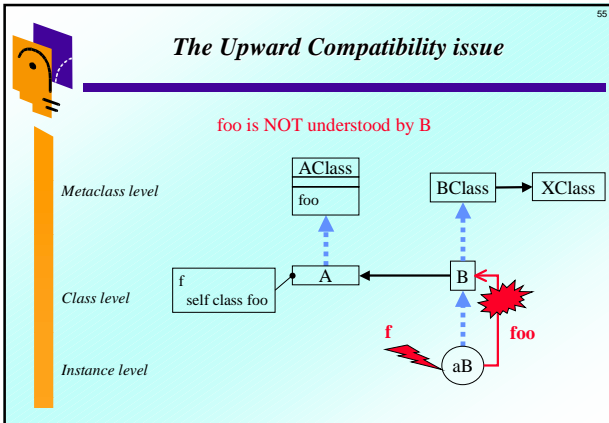





- ### Need for Metaclass Composition
- **Why?**
    - Reuse
    - A class can have many properties
  - **How?**
    - Mixins at the metaclass level
  - **Problems to deal with?**
    - Compatibility



- ### Examples of inter-level communication
- `Object>>printOn: aStream`  
`[title]`  
`title := self class name.`  
`aStream nextPutAll: ...`
  - `Collection class>>with: anObject`  
`[newCollection]`  
`newCollection := self new: 1.`  
`newCollection add: anObject...`



61




---

*Conclusion*

62

*Summary - 1*

---


- **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)

63

*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**



64

*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

65

*Thanks for your attention*  
*Questions? Comments?*

---



**MetaclassTalk**  
Reflection & Meta-Programming  
powered by Smalltalk

Documents & Download  
<http://csl.ensm-douai.fr/MetaclassTalk>