The Design and Implementation of Guaraná

http://www.dcc.unicamp.br/~oliva/guarana/

Alexandre Oliva — oliva@dcc.unicamp.br

Luiz Eduardo Buzato — buzato@dcc.unicamp.br

Laboratório de Sistemas Distribuídos

Instituto de Computação

Universidade Estadual de Campinas

May, 1999

Summary

- Introduction
- Approaches to Reflection
- **Guaraná**
  - Reflective Architecture
  - Implementation
- Conclusion

Introduction

- Object Orientation
  - Interface
  - Implementation
- Open Architectures
  - Adaptation
- Computational Reflection
  - Structural meta-information
  - Separation in levels
    - Base: functional
    - Meta: management

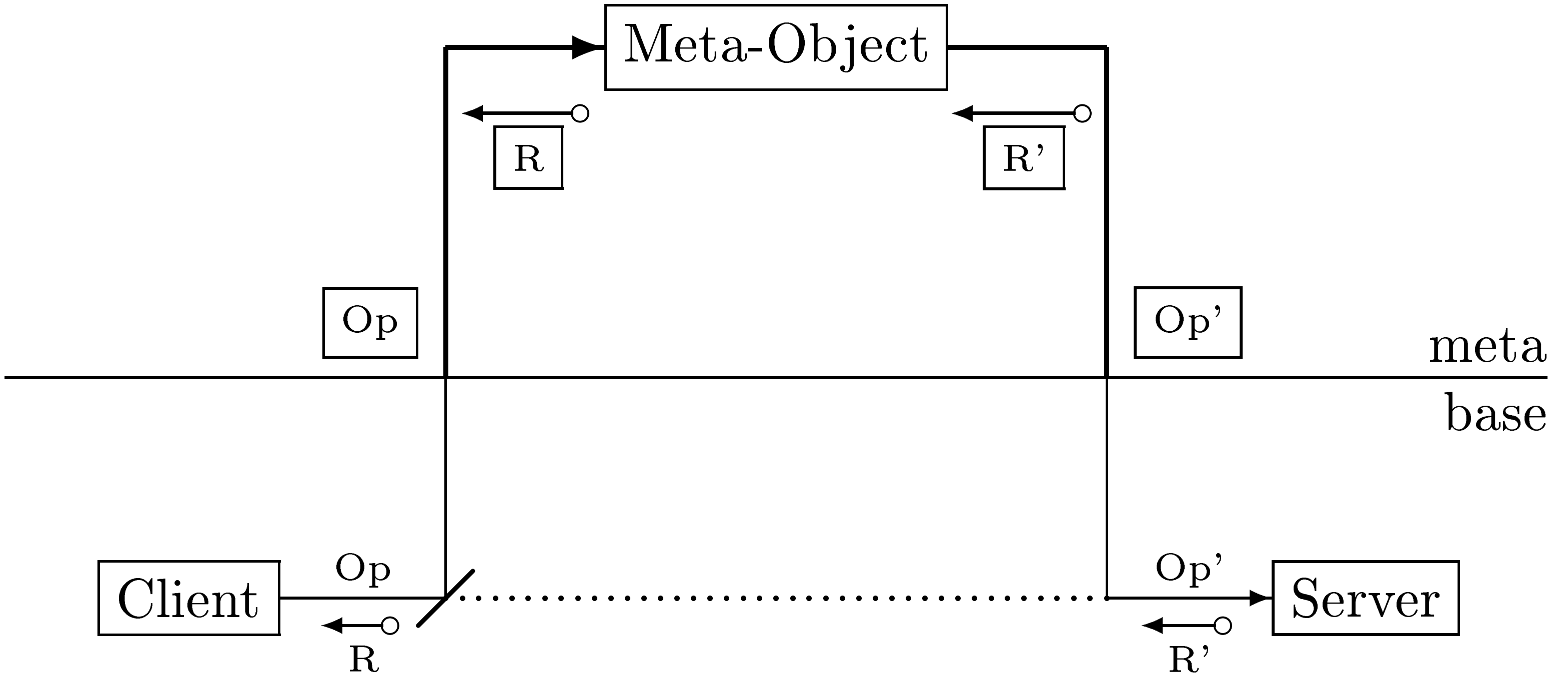
Related Work

Approaches to Reflection

- Compile-time reflection
  - little flexibility
- Meta-circular interpreters
  - performance penalty
- Heterogeneous meta-objects
  - conflicting extensions
- Homogeneous meta-objects
  - interception

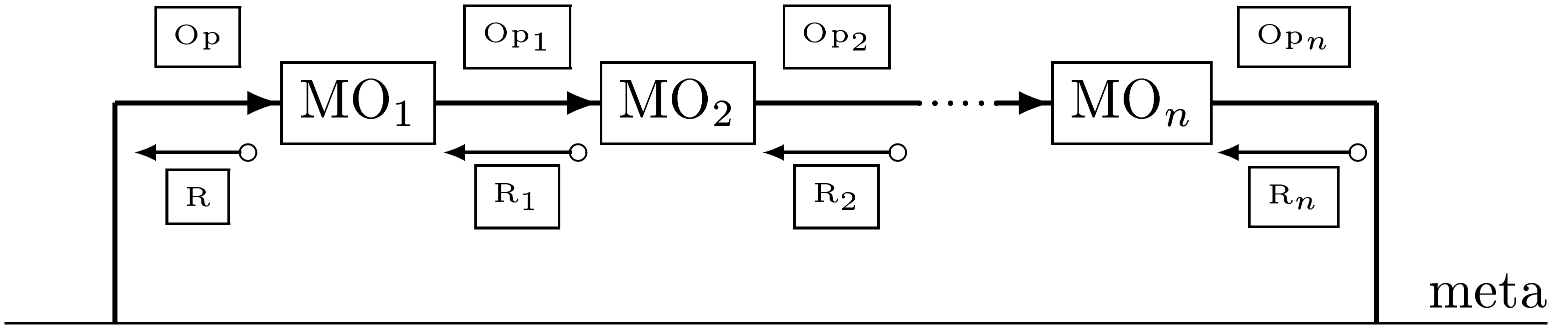
Related Work

Interception Model



Related Work

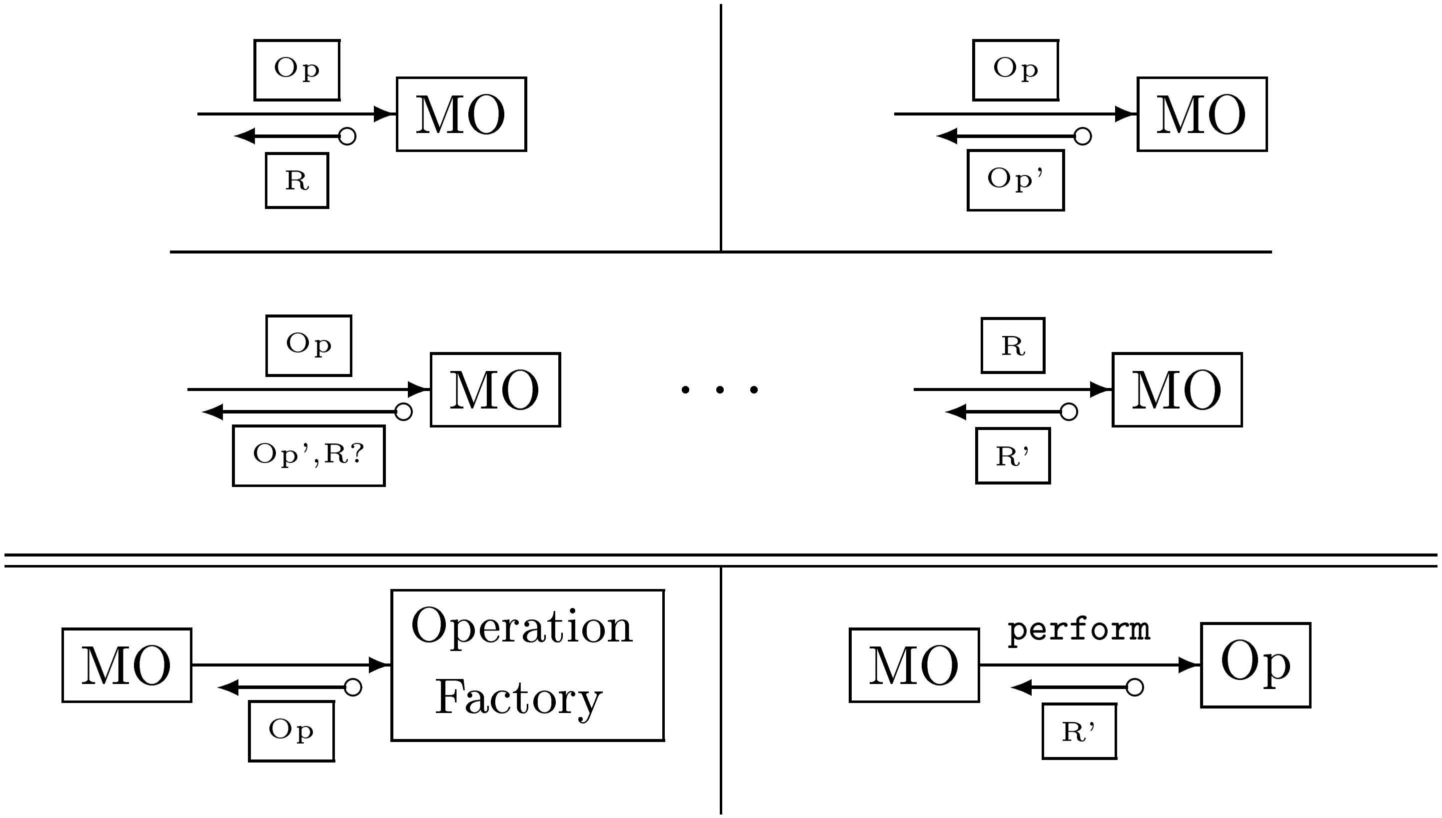
Chain of Responsibility



- Intrusiveness
- Non-concurrency
- Nested execution
- Difficulty to resolve conflicts

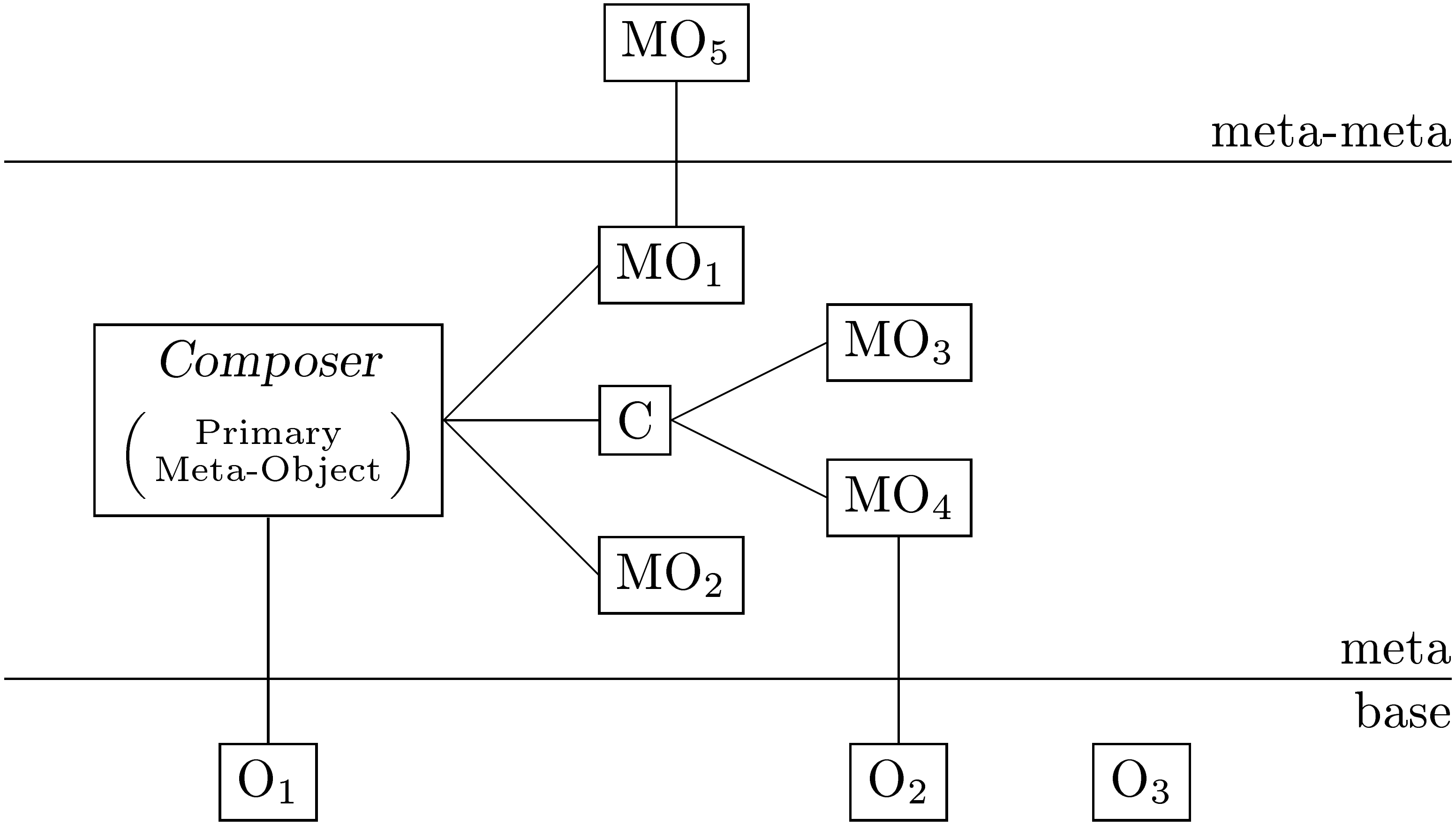
Guaraná: Reflective Architecture

Operations and Results



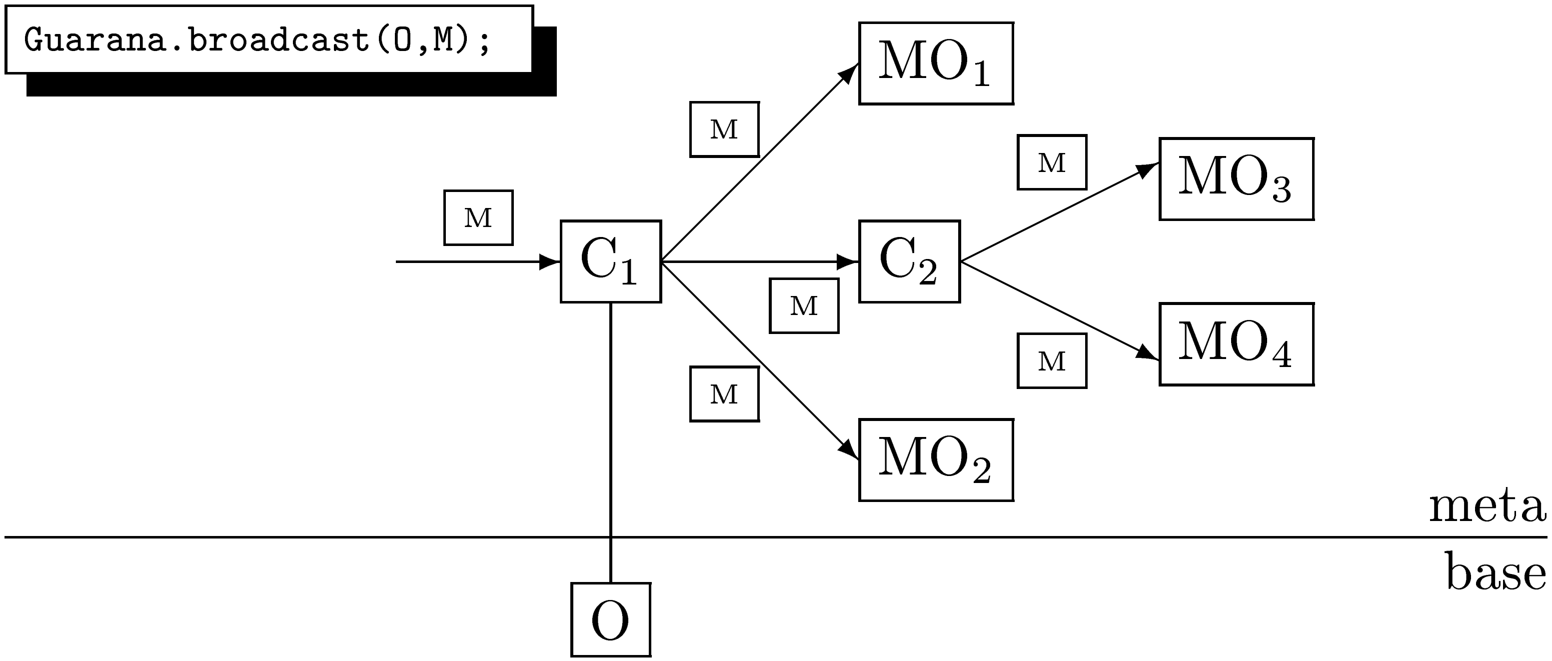
Guaraná: Reflective Architecture

Composite Meta-Configuration



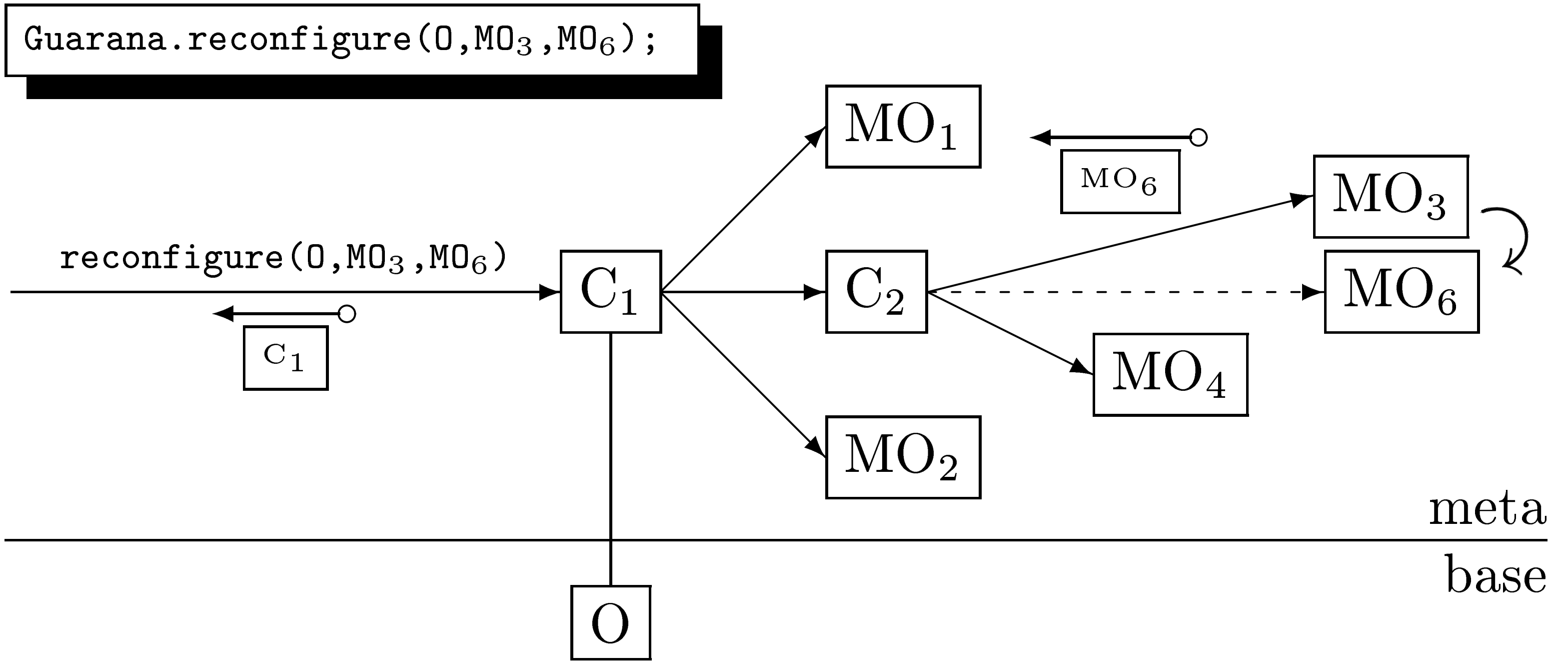
Guaraná: Reflective Architecture

Message Broadcasting



Guaraná: Reflective Architecture

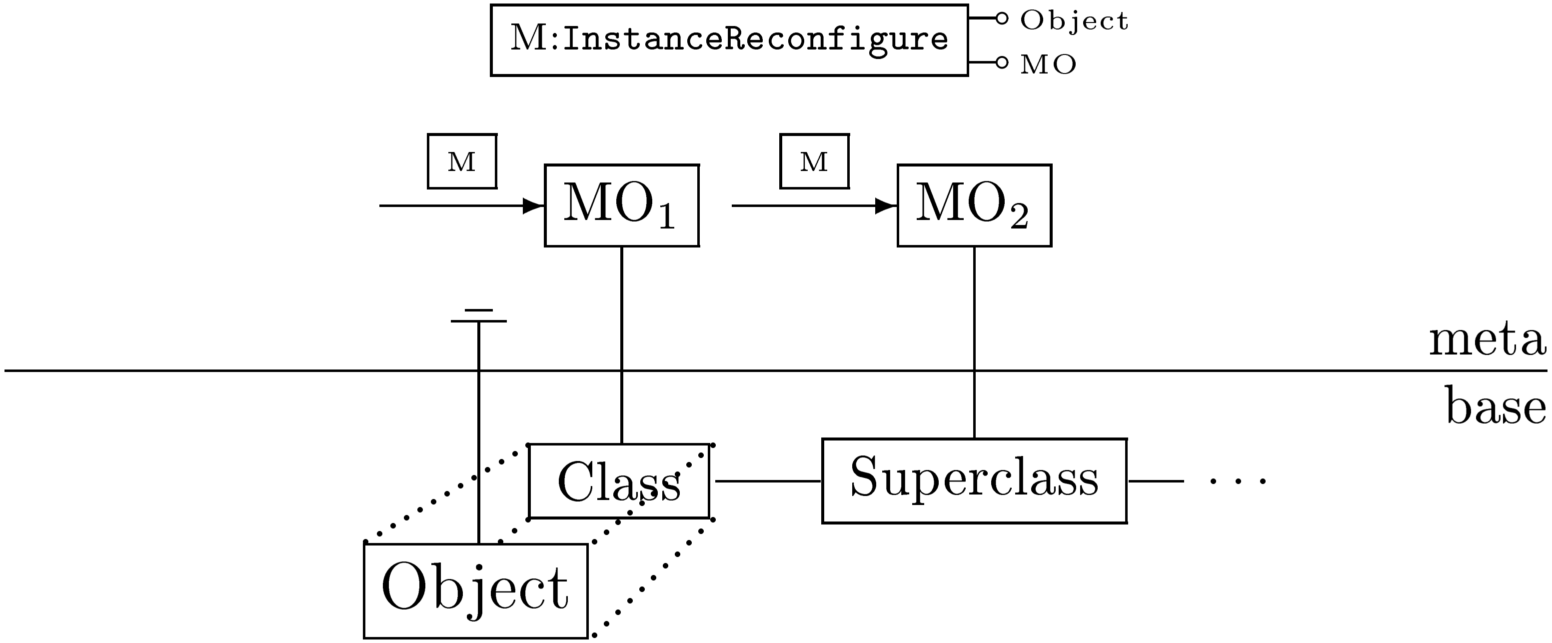
Dynamic Reconfiguration



Guaraná: Reflective Architecture

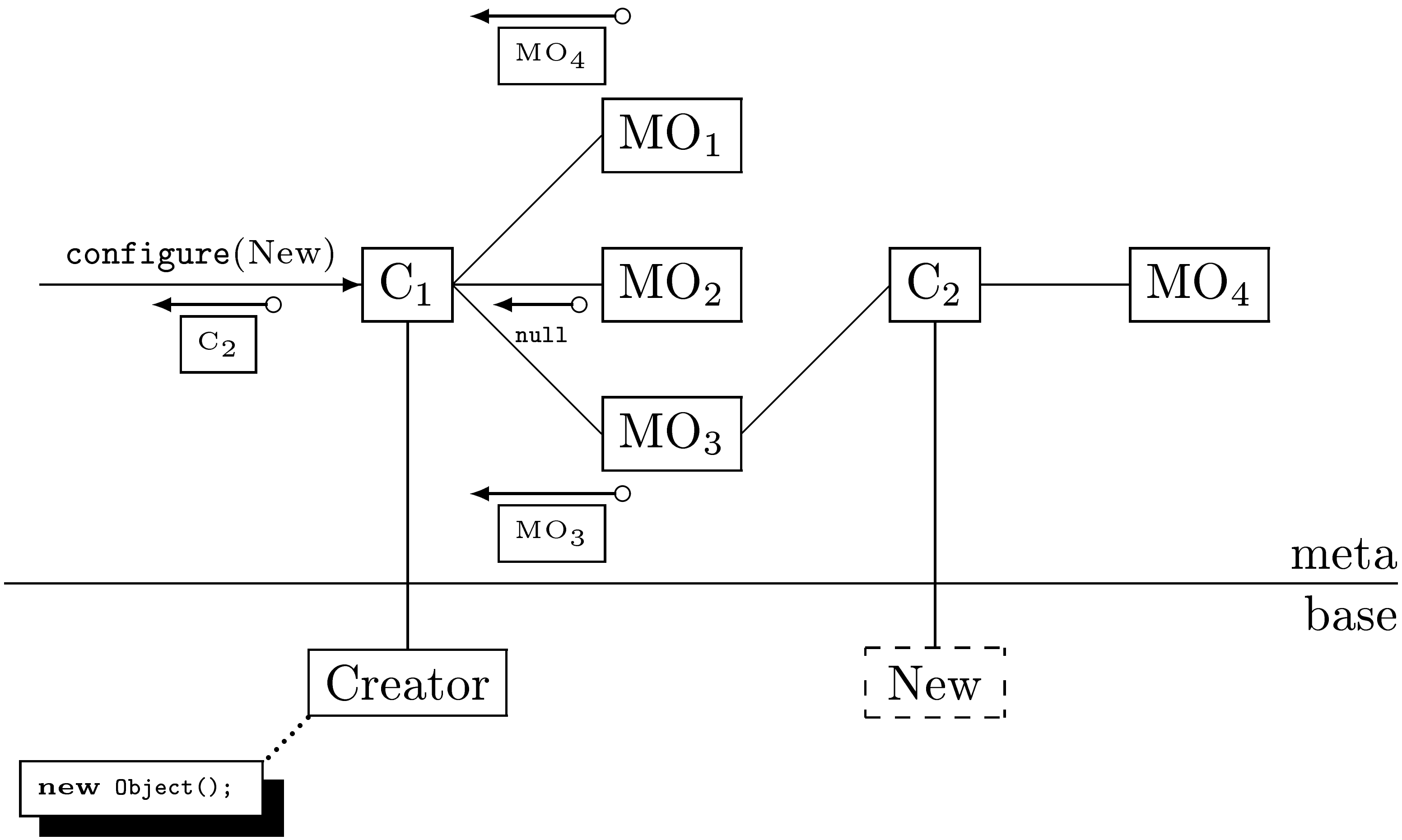
Dynamic Reconfiguration (2)

```
Guarana.reconfigure(Object,null,MO);
```



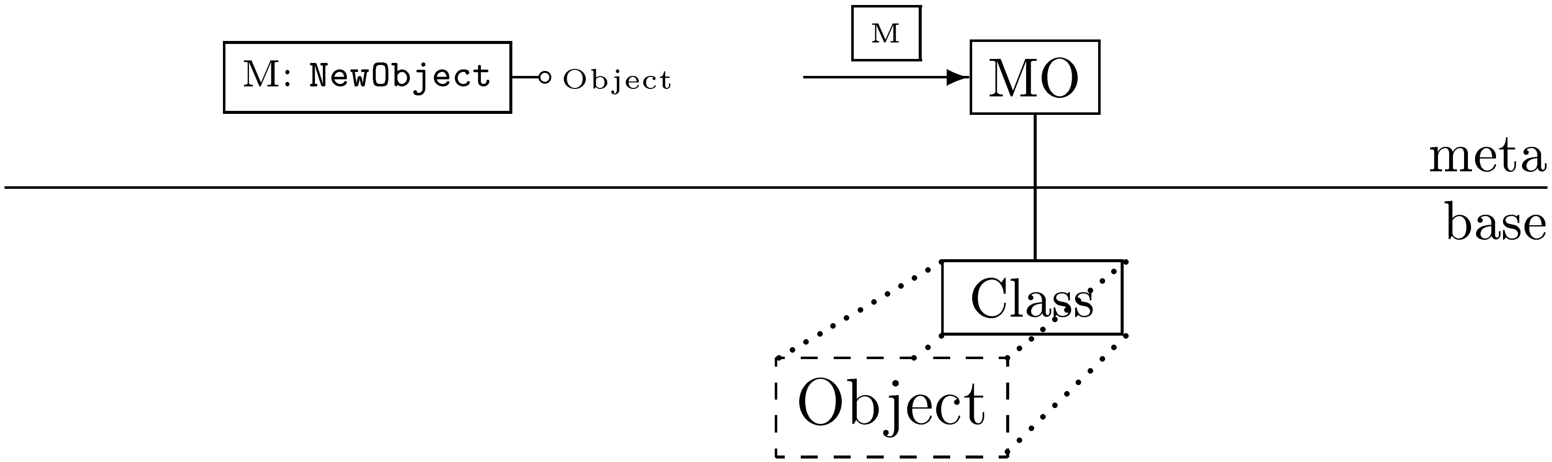
Guaraná: Reflective Architecture

Meta-Configuration Propagation

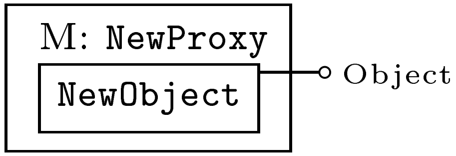


Guaraná: Reflective Architecture

Meta-configuration Propagation (2)




```
Object = Guarana.makeProxy(Class,MOp);
```



. . .

```
reconfigure(Object,null,MOp);
```

Guaraná: Reflective Architecture

Security

- Composition hierarchy
  - Limiting abilities of components
  - Restricting operation creation
  - Preventing reconfigurations
  - Denying propagation
- Inaccessible meta-configuration

Guaraná: Implementation

Why not 100% Pure Java?

- Java interpreter in Java
  - Performance
  - Native methods
- Bytecode preprocessor
  - Verifier
  - Native methods
  - Standard classes

Guaraná: Implementation

Extending Java^{TM}

- Java Programming Language
  - No compiler or bytecode modification
  - Pre-compiled applications
- Java Reflection API
  - Class-based access control
  - No interception
- Java Virtual Machine
  - Localized changes
  - Interception penalty
  - Performance impact on non-reflective code

Guaraná: Implementation

JIT-compiled `getfield`

load	r2 $\leftarrow$ r1[MO_offset]		; *r1 points to an object*
spill	r2, maybe r1		; *spill all dirty registers*
branch	intercept	**if** r2 $\neq$ **null**	; *is the object reflective?*
load	r2 $\leftarrow$ r1[fld_offset]		; *load field value*
spill	r2		; *end of basic block*
goto	endif		; *do not intercept*
intercept:			
reload	r1 and r2		; *reload object and meta-object*
call	intercept(r2, r1, fld_id)		; *perform interception*
spill	result		; *end of basic block*
endif:			
reload	r1 and r2		; *reload object and field value*

Conclusion

- Decomposition
  - Implementation
  - Debugging
  - Maintainance
- Reuse
- Dynamic reconfiguration
- Flexibility $\times$ Security
- Acceptable performance?

Future work

- **MOLDS**: Meta-Object Library for Distributed Systems
  - Persistence functionality already implemented
- Reflection for 100% Pure Java ORBs

Obtaining Guaraná

- Free distribution (GNU GPL)
- Extensive documentation

`http://www.dcc.unicamp.br/~oliva/guarana/`