

# Virtual Smalltalk Images: Model and Applications

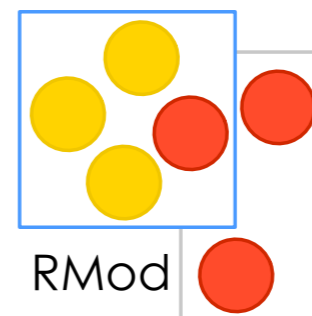
Guillermo Polito, Stéphane Ducasse, Luc Fabrese,  
Noury Bouraqadi



Université Lille Nord de France  
Pôle de Recherche  
et d'Enseignement Supérieur



*Inria*



Université  
Lille1  
Sciences et Technologies

# The context: Pharo

- Object oriented language
- Dynamic typing
- Highly reflective
- Image based
- Metacircular

# Going meta has limitations <sup>[1,2]</sup>

## Metacircularities

- hidden and uncontrollable dependencies e.g.,
  - core libraries (e.g., `#do: iteration method`)
  - core tools (e.g., compiler, debugger)
- Once broken, they are not easily reconstructed
- VM still needed for many tasks (e.g., process manipulation)

[1] Bootstrapping reflective languages: the case of Pharo, Polito et al. 2012 (in submission)

[2] The meta in meta-object architectures, Denker et al. 2008

# Some existing solutions

- Casaccio et al. 's object spaces[3] allowed atomic operations on classes
- Bootstrapping a Smalltalk[1] allowed the recreation of an entire Smalltalk system to alter metacircularities
- Pharo's emergency evaluator allows recovery of some critical errors
- Debugging/modifying the virtual machine to have complete control
- Changes file allows code recovery
- Ad-hoc solutions (copy the class, modify copy, replace...)

[1] Bootstrapping reflective languages: the case of Pharo, Polito et al. 2012 (in submission)

[3] Object spaces for safe image surgery, Casaccio et al. 2009

# What we would like from a reflective system

- Full control
- Atomicity of changes
- Interactive
- High level tools for system recovery
- Recovery of both code and application data

# Our solution: Virtual images with Object Spaces

- An **full** image is contained and controlled by another image
- The container image is the **host**
- The contained image is the **guest**
- An **Object Space**[1] reifies the guest image and allows the host to manipulate it as an object

[1] Gwenaël Casaccio, Damien Pollet, Marcus Denker, and Stéphane Ducasse. Object spaces for safe image surgery. In Proceedings of ESUG International Workshop on Smalltalk Technologies (IWST'09)



# Object Spaces

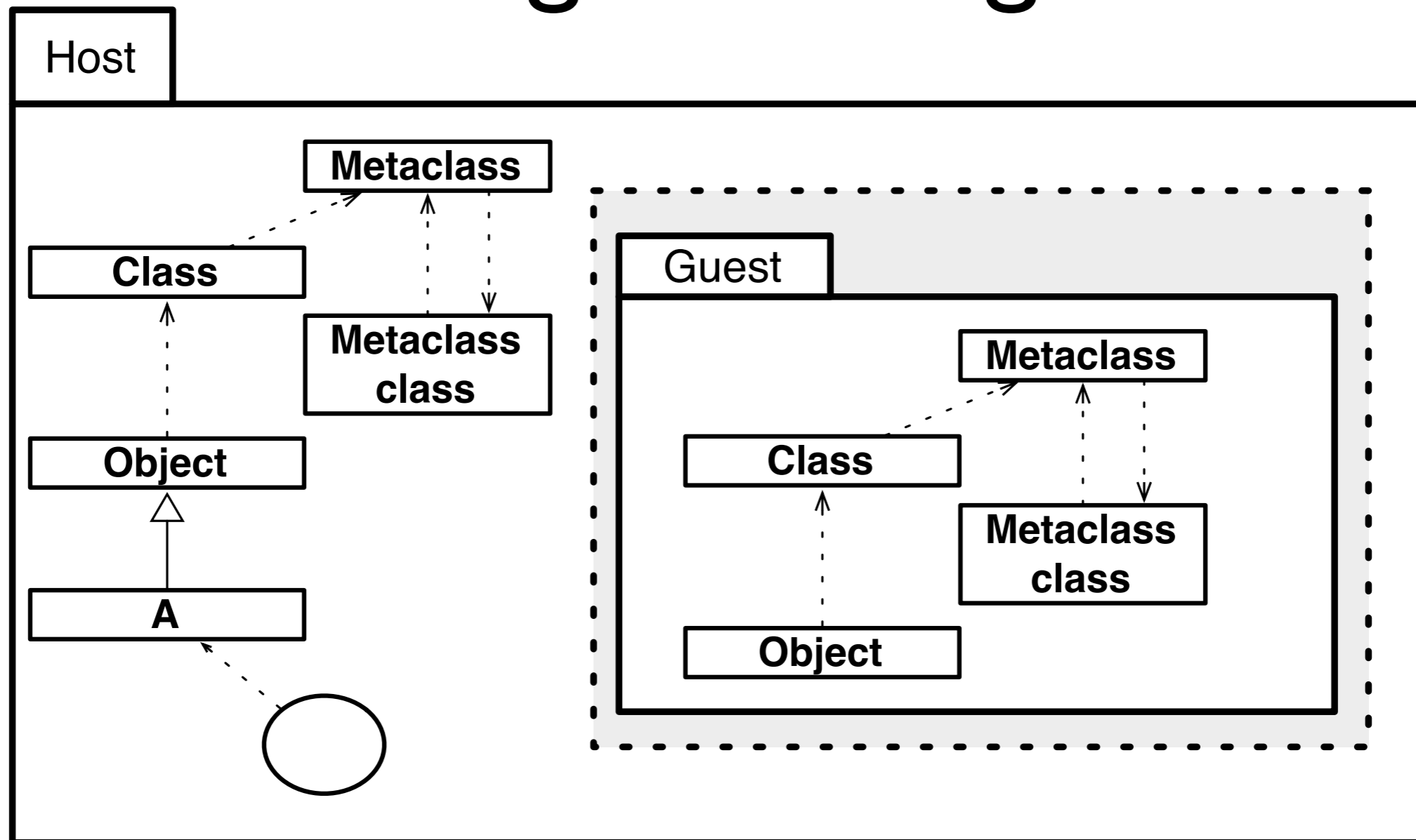
# Oz Object Spaces the model

A reification of a Smalltalk image with two main components:

- The **guest image** it contains and encapsulates
- A **membrane** of objects controlling the communication between host and guest objects



# Oz Object Spaces the guest image



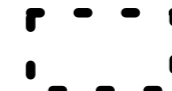
Caption:



Smalltalk Image

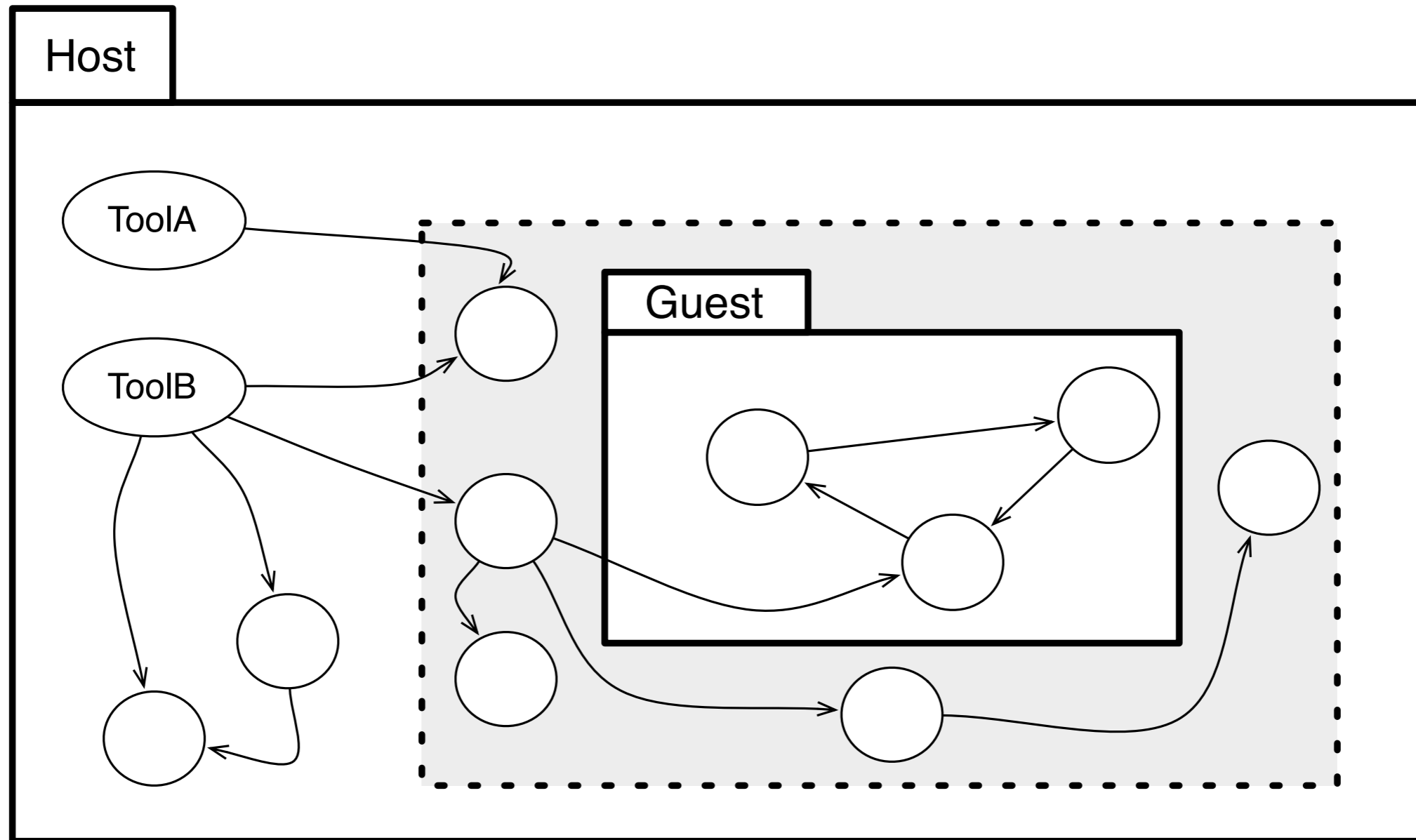


Object



Object Space

# Oz Object Spaces the membrane



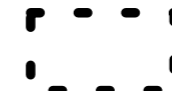
**Caption:**



Smalltalk Image



Object



Object Space

## Oz Object Spaces membrane: the façade

- The object space façade is the concrete reification of the guest image
- Access to well known objects in the object space. e.g., nil, true and false
- Access to classes in the object space
- Conversion of literals from the host to the guest, and viceversa

# Oz Object Spaces membrane: mirrors

- Direct object manipulation
  - Field manipulation: get and set values from object fields
  - Size calculation: get the size of objects
  - Class access: introspect and modify the class of an object
  - Testing and conversion operations. e.g., `#isNil`, `#asBoolean`
- Some mirrors provide specialized operations. e.g., a `ClassMirror` supports the `#basicNew` operation

## Oz Object Spaces control of execution

- Full control of an object space processes via mirrors i.e., create, suspend, resume and terminate processes and alter values of their associated contexts
- Provide controlled windows of time for execution to an object space

## Oz Object Spaces control of communication

- One way communication: host to guest
- Communication is achieved by creating and injecting objects into the object space
- Mirrors ensure that object injection is safe and fulfills the transitive closure property
- Sending messages: the particular case of injecting a process with the desired code and introspect its result after finished
- Literal translation from and to guest and host

# Oz Object Spaces applications

- Image surgery
- Image recovery and better Emergency Kernel
- Controlled interruption
- Image side Stack overflow and infinite loop detection
- Sandboxing

# The Vision

- Many versions of the system running side by side
- Controlling resources at the language level
- Secure
- Completely modifiable and adaptable (while running)
- Specialized for particular needs
- With tools to analyze, modify and fix a system
  - System surgeon
  - System browsing and debugging



# Virtual Smalltalk Images: Model and Applications

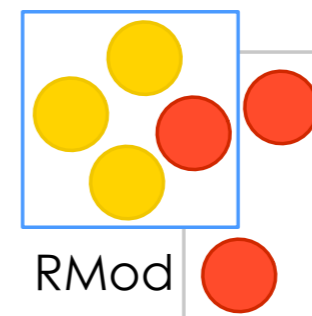
- Oz Object spaces reify full Pharo images
- Irrestricted manipulation of guest's objects through mirrors
- Control of guest's execution



Université Lille Nord de France  
Pôle de Recherche  
et d'Enseignement Supérieur



*Inria*



Université  
Lille1  
Sciences et Technologies

# Behind Oz Object Spaces: the implementation

# Oz Object Spaces the implementation

We implemented Oz on top of Pharo20:

- Virtual machine support for multiple images
- Image side library to install in the host (implementing the membrane objects)

# Oz Object Spaces

## Challenging infrastructure

- **Special Objects Array:** an array the VM uses to understand the running image objects i.e., the nil, true and false instances, the compact classes, etc
- **Green Threads:** concurrency is managed with reified user-level threads, the Process instances. No real multithreading is supported
- **VM Single Interpreter Assumptions:** global variables all around, plugin are single-interpreter aware

# Oz Object Spaces

## memory layout

Host and Guest images share the same heap

The Object Memory									
nil	false	true	...	nil	false	true	...	'hi!'	...
host	host	host		guest	guest	guest		host	

- Reuse of VM memory mechanisms
- No extra VM support to handle cross-image references
- Shared garbage collection

# Oz Object Spaces mirror implementation

Mirrors manipulate objects through two main primitives:

- Execute a method on an object
- Try a primitive on an object

Mirror hierarchy based on roles and internal structure

- ObjectMirror, ByteObjectMirror, WordObjectMirror
- ClassMirror, MethodDictionaryMirror, ProcessMirror

# Oz Object Spaces

## process manipulation

- Processes are simple objects
- In the host, they are represented through `ProcessMirror`, `ProcessSchedulerMirror` and `ContextMirror`
- Mirrors allow manipulation and maintain consistency of the process model

# Oz Object Spaces context switch

Execution of code inside an object space is based on a context switch:

1. The VM installs the object space as the current environment to run
2. Runs until being preempted
3. Gives control back to the host



# Oz Object Spaces

## Pharo image contract

An image contract is composed by:

- **Services:** reachable objects from the image's root. e.g., nil, true, false, the SystemDictionary. They appear in the *special objects array* (the root object of an image).
- **Format of the objects:** amount and meaning of the instance variables of core objects. They are configured in the membrane.

# Virtual Smalltalk Images: Model and Applications

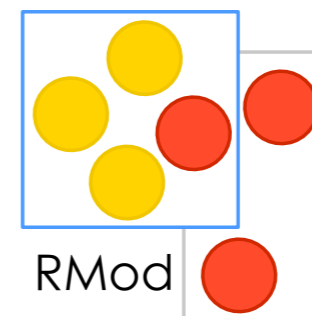
- Oz Object spaces reify full Pharo images
- Irrestricted manipulation of guest's objects through mirrors
- Control of guest's execution



Université Lille Nord de France  
Pôle de Recherche  
et d'Enseignement Supérieur



*Inria*



Université  
Lille1  
Sciences et Technologies