Partial Behavioral Reflection:
Spatial and Temporal Selection of Reification

Éric Tanter
DCC/CWR, University of Chile
OBASCO Project, École des Mines de Nantes – INRIA

OOPSLA 2003, Anaheim, California, USA

Joint work with:

- **Jacques Noyé** – INRIA/EMN,
- **Denis Caromel** – INRIA/I3S/U.Nice/IUF,
- **Pierre Cointe** – EMN

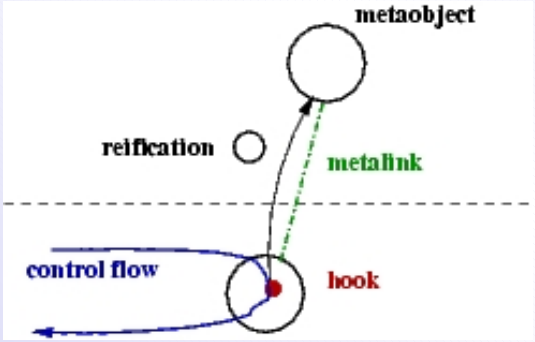
General Context

- **Reflection** is a powerful framework for dealing with *open systems* and *separation of concerns* (SOC)
- Impact remains somehow marginal because of
  - **efficiency**
  - **complexity/usability** (granularity, scope, composition)
- Main trend AOP addresses these issues by *restricting dynamicity and/or expressiveness*
- Alternative way $\longrightarrow$ better understand:
  - range of possibilities
  - **continuum between Reflection and SOC** (esp. AOP)

Specific Context

Behavioral Reflection and Runtime Metaobject Protocols

metaobjects *reasoning and acting upon* **reifications**
of a program described in terms of **operations**. [McAffer:Reflection96]



Issues of behavioral reflection

- **cost of reification**
- **view on metalink** limits usability and increases complexity
- **adequate MOP design** depends on applications

Cost of Reification

A message send in a method body:

```
A a = o.foo(5);
```

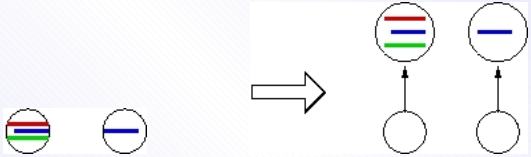
is reified:

```
Object[] args = new Object[]{ new Integer(5) };
Method m = o.getClass().getDeclaredMethod("foo");
Object result = metaobject.handle(this, o, m, args);
A a = (A) result;
```

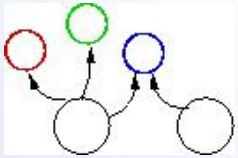
$\longrightarrow$ *avoid useless reification*

Classical View on Metalink

- The metalink is **entity-based**: per object, per class
  → *leads to* tangled *metalevel*



- Not feasible/easy to setup a **concern-based** metalevel decomposition



→ *flexible metalink (granularity, cardinality)*

MOP Design

MOP design is driven by a **trade-off** between

- expressiveness
- flexibility
- efficiency

Problem is

- appropriate trade-off depends on **expected applications**
- this trade-off is **frozen** in existing reflective systems

$\longrightarrow$ *adjustable trade-off*

Our Contributions

1. performance: *selective reification*
2. metalink: *hooksets*
3. MOP design: *open MOP support*
4. Java incarnation: **Reflex**

Selective Reification: spatial and temporal selection

Spatial Selection

what to reify?

- entity selection
- operation selection
- intra-operation selection

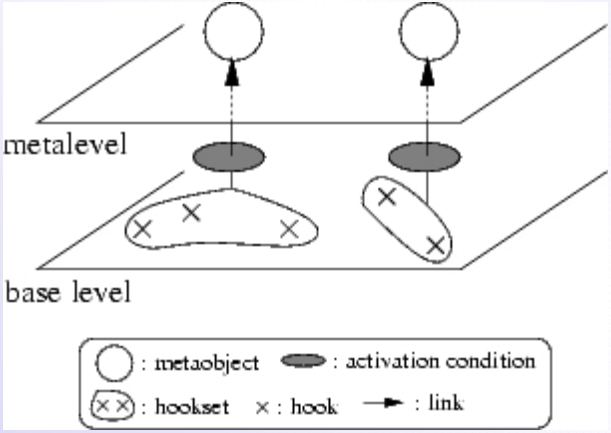
Temporal Selection

when to reify?

- hook activation subject to a predicate (dynamic conditions)

Hookset Model

- selective reification → precise hooks
- how to group and manage them? → **hooksets**



Hooksets

- a *named* set of hooks
- may gather hooks scattered in various objects

  $\longrightarrow$ *crosscutting metaobjects*
- an object may be involved in several hooksets

  $\longrightarrow$ *better modularity of metalevel*
- composable

  $\longrightarrow$ *reuse*

Metalink

- link *hookset→metaobject*
- described by several attributes
  - **scope** (object, class, hookset)
  - **activation** (condition)
  - **control** (before, after, replace)
  - ...

Model

- spatial selection: **hookset definitions**
- temporal selection: **activation layer**
- similar to *Event–Condition–Action* model

Definition

Separate definition of *hooksets*, *metaobjects*, *links*

- different concerns
- roles: *metaprogrammer*, *assembler*

Open MOP Support

Metalevel architect defines **specific MOPs** for **specific needs**.

Give him the possibility to specify:

- *operations*
  - what is to be considered an operation
  - what operations are supported
- *metaobjects*
  - what is the interface of metaobjects
  - which data is actually reified, how it is passed

Several MOPs can **coexist** within a given application.

Reflex: A Java Framework for PBR

- **Portable** Java library (bytecode transformation [Javassist])
- Java specificities:
  - class-based, single inheritance, strongly typed
  - classes loaded dynamically, frozen when loaded
- main **restrictions** to our generic model:
  - changes to hooksets do not affect already loaded classes
  - activation conditions are always checked

Reflex package = Core Reflex + (optional) Standard MOP

Core Reflex

- defining **MOPs**
  - operation classes / metaobject interfaces / hook installers
- defining **hooksets** as a set of (composed) **primitive hooksets**
  - a primitive hookset is *operation-specific*
  - characterized by class and operation selectors
    $\longrightarrow$ *intentional and expressive (not purely syntactic)*
- defining **links**
  - hooksets / attributes / metaobject definition
- **static** and **dynamic** configuration of reflective needs

Architect API

```
defineOperationSupport(...)
setDefaultControl(...), setDefaultScope...
```

Assembler API

```
defineHookset(...)
undefineHookset(...)
getHookset(...)
defineLink(...)
undefineLink(...)
getLink(...)
```

Programmer API

```
setMetaobject(...)
getMetaobject(...)
setActive(...)
getActive(...)
createObject(...)
```

- Architect and Assembler APIs are accessible statically
- All APIs are accessible dynamically

```
ReflexAPI.getArchitect()
ReflexAPI.getAssembler()
ReflexAPI.getProgrammer()
```

Standard MOP

- ready-to-use
- expressive:
  - message send/receive, cast, creation, ...
  - all available information is reified

Examples (in the paper)

- **Observer pattern**
  - good modularity properties (esp. locality and pluggability)
  - pure OO
- **Transparent futures**
  - cast control
  - activation
  - expressive/customizable selection framework

Conclusion

Main objective: *enhance behavioral reflection applicability*

- **Model of hooksets**
  - *generalizing the classical view on metalinks*
  - *support for crosscutting metaobjects*
- **Selective reification**
  - *spatial and temporal dimensions*
  - *intentional and expressive description of MOP entry points*
- **Open MOP support**
  - *specialized MOPs for specific needs*
- **Reflex for Java**

Perspectives

Core Reflex

- *enhancements, extensions, and optimization*
- *Reflex as a MOP generator*

Applications

- MOP specialization: *concurrency model*
- distributed systems: *context-aware / adaptable applications*

Reflection and AOP

- strength of AOP lies in *language support*,
  in particular Aspect-**Specific** Languages (ASLs)

> AOP is *problem-oriented*
> Reflection is *solution-oriented*

- but ASLs complicate aspect interaction & composition
  $\longrightarrow$ *shift to* generic *approaches*

Partial Behavioral Reflection and generic AOP

(AspectJ terminology)

- pointcuts:
  - static $\longrightarrow$ *hooksets*
  - dynamic $\longrightarrow$ *link activation*
- advice kind $\longrightarrow$ *link control*
- advice $\longrightarrow$ *metaobject body*

On-Going and Future Work

- Integration of **Event-based AOP** over Reflex
- **Multiple language support** for configuring Reflex
  - Expressive language for defining hooksets and links

    $\longrightarrow$ *raise the level of abstraction*
  - ASLs on top of this generic reflective infrastructure

    $\longrightarrow$ *provide guarantees, lower complexity (*hide the meta*)*

Questions?