

# Object-oriented Encapsulation for Dynamically Typed Languages

Nathanael Schärli  
*Software Composition Group, University of Berne, Switzerland*  
Andrew P. Black  
*OGI School of Science and Engineering, Portland, USA*  
Stéphane Ducasse  
*Software Composition Group, University of Berne, Switzerland*

## Why Another Approach?

1. Lack of flexibility
  - ❑ encapsulation policies for fixed categories of clients
2. Lack of expressiveness
  - ❑ No object-level encapsulation
3. Dependence on static type systems
  - ❑ Semantics is based on static types

November 30, 2004

Object-Oriented Encapsulation

2

## Illustrating Example: Class Morph

- **Morph** is the root of all graphical objects in the Squeak (Smalltalk) UI framework



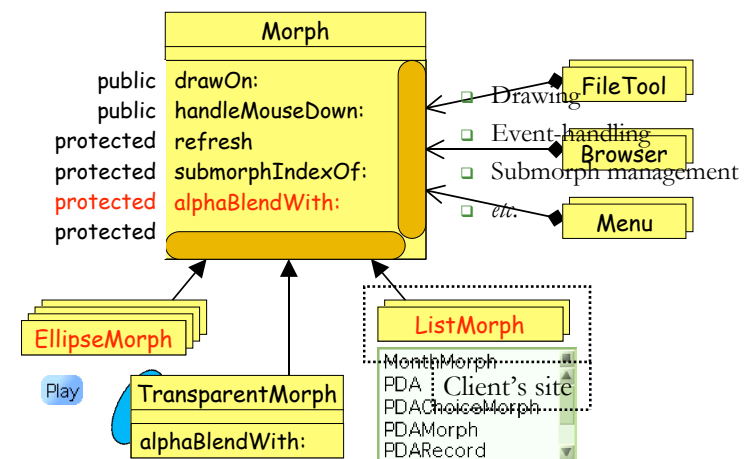
- Code in Smalltalk syntax

November 30, 2004

Object-Oriented Encapsulation

3

## P1. Lack of Flexibility



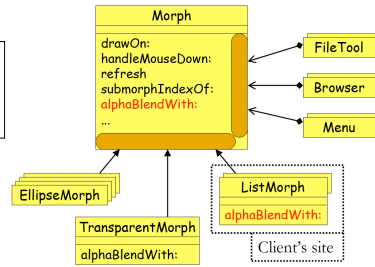
November 30, 2004

Object-Oriented Encapsulation

4

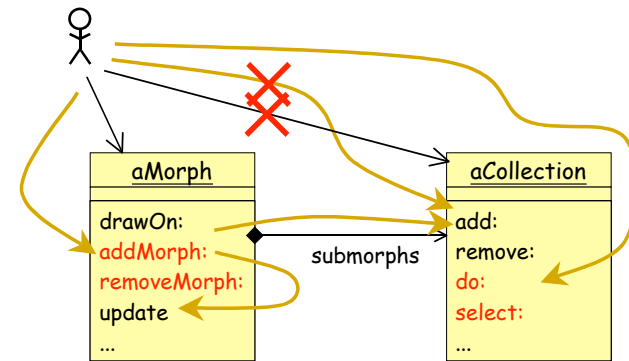
## Summary

Unnecessarily wide and error-prone interfaces



- Fragility with respect to changes
  - Adding a method can *break* any existing subclass

## P2. No Object-level Encapsulation

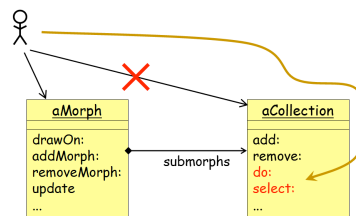


Enumeration protocol (>50 methods)

## Summary

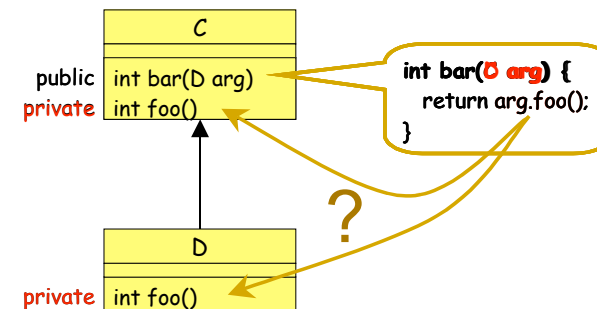
Subobjects are either

- *fully accessible* or
- *completely inaccessible*

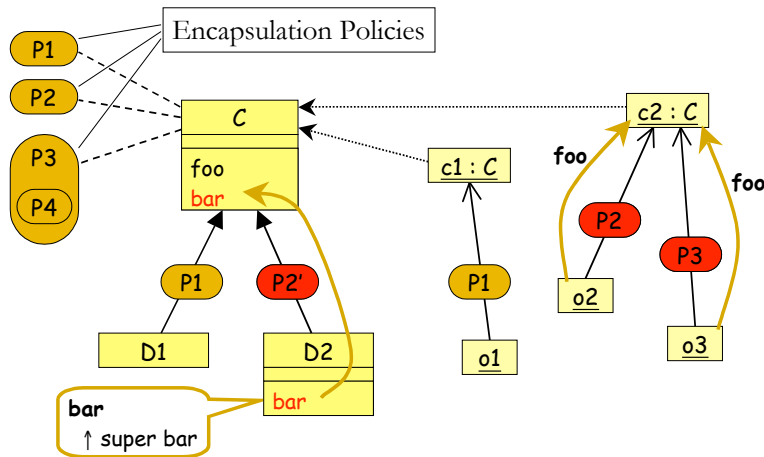


- Workarounds are clumsy and error prone
- Difficult to implement data structures that are both *safe* and *convenient* to use

## P3. Semantics Based On Static Types



## Our Proposal in a Nutshell



November 30, 2004

Object-Oriented Encapsulation

9

## What is an Encapsulation Policy?

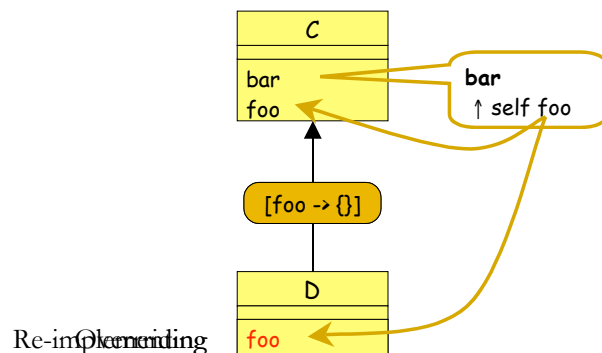
- Mapping from method selectors to access rights
  - Two access rights
    - The right **c** to *call* a method
    - The right **o** to *override* a method
- Example
  - $[\text{foo} \rightarrow \{\mathbf{co}\}, \text{bar} \rightarrow \{\mathbf{c}\}, \text{check} \rightarrow \{\mathbf{o}\}]$
- For convenience
  - $[\text{foo}, \text{bar}] \Leftrightarrow [\text{foo} \rightarrow \{\mathbf{co}\}, \text{bar} \rightarrow \{\mathbf{co}\}]$

November 30, 2004

Object-Oriented Encapsulation

10

## Overriding vs. Re-implementing



Re-implementing

November 30, 2004

Object-Oriented Encapsulation

11

## Encapsulation Semantics

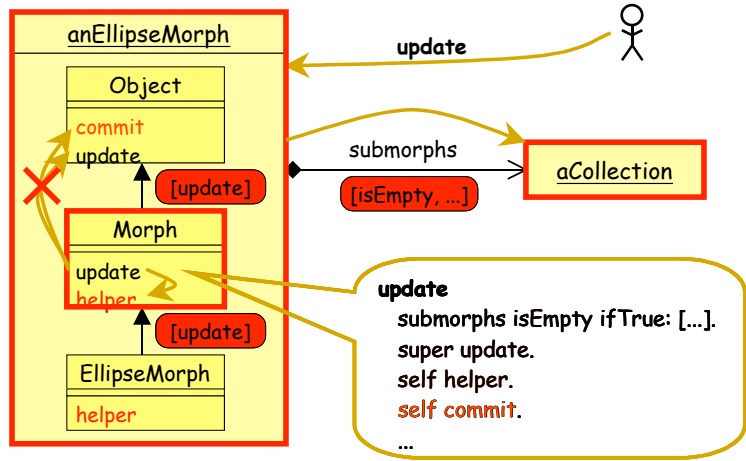
- Encapsulation affects only message sends
  - State is completely inaccessible from the outside
- We distinguish between three *syntactically* different kinds of message sends
  - self-sends (**self** foo)
  - super-sends (**super** foo)
  - object-sends (**anObject** foo)
- We explain the semantics on an example...

November 30, 2004

Object-Oriented Encapsulation

12

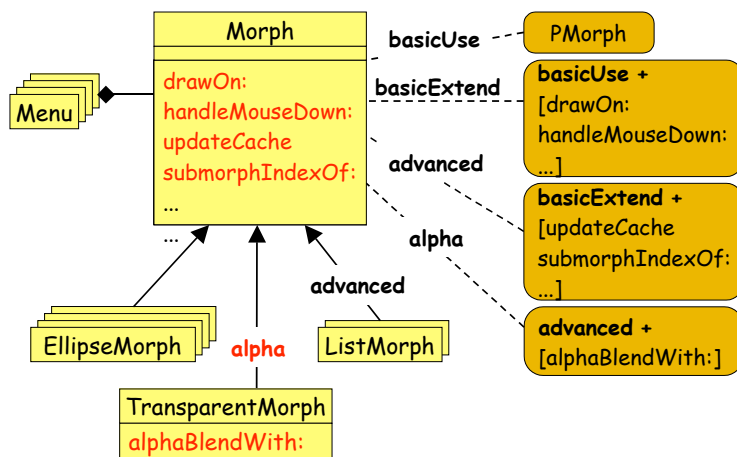
## Message Semantics



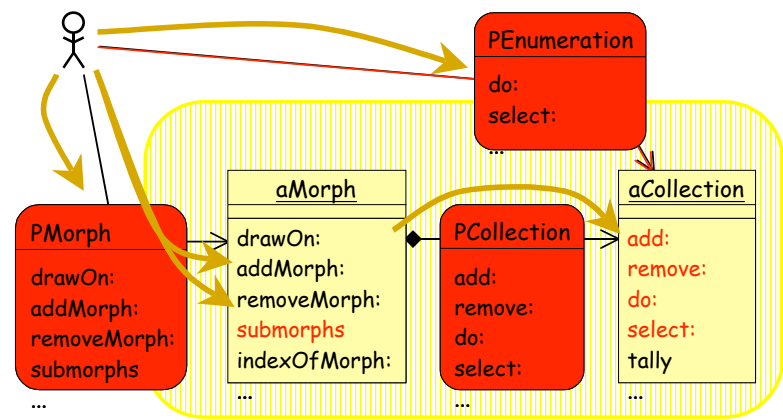
## Message Semantics: Summary

- Valid messages
  - Object-sends: receiver is treated as a black-box  
Policy associated with the *receiver reference* is relevant
  - Self-sends and super-sends: always sent to current object  
Policies used in the *inheritance hierarchy* are relevant
- Message lookup
  - Object-sends and super-sends use traditional message lookup
  - Self-sends take into account whether a method is *overridden* or *re-implemented*

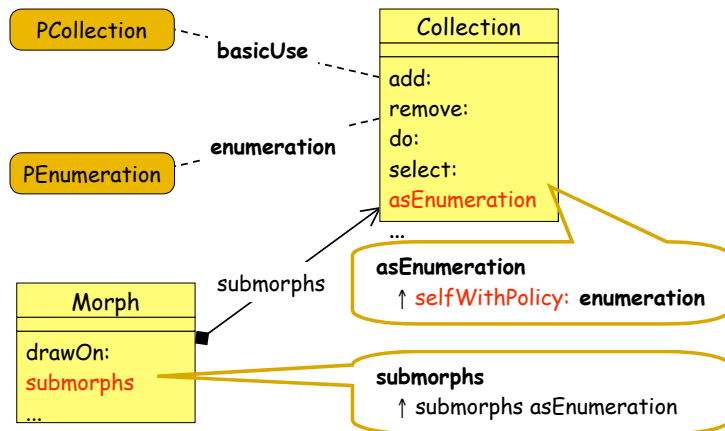
## Making the Class Hierarchy Robust



## Controlling Access to Sub-objects



## Controlling Access to Sub-objects (2)



November 30, 2004

Object-Oriented Encapsulation

17

## Evaluation Against the Problems

1. Lack of flexibility
  - ❑ encapsulation policies are *reusable* and *composable*
  - ❑ *Unlimited* number of encapsulation policies
  - ❑ Clients can *select* and *customize* encapsulation policies
2. Lack of expressiveness
  - ❑ Supports both *class-level* and *object-level* encapsulation in a *uniform* way
3. Dependence on static type systems
  - ❑ Semantics is based on different forms of message sends

November 30, 2004

Object-Oriented Encapsulation

18

## Implementation

- Prototype implementation in Squeak (Smalltalk)
  - ❑ Distinction between message sends is *syntactic*:  
Class-level encapsulation at compile-time
  - ❑ Per-object (reference) encapsulation policies:  
Required VM modifications
  - ❑ Early alpha-version:  
Extensive use of per-object (reference) encapsulation  
< 15% performance penalty

November 30, 2004

Object-Oriented Encapsulation

19