Studying a Minimal Object-Oriented Kernel

Classes as Objects?

“The difference between classes and objects has been repeatedly emphasized. In the view presented here, these concepts belong to different worlds: the program text only contains classes; at run-time, only objects exist. This is not the only approach. One of the subcultures of object-oriented programming, influenced by Lisp and exemplified by Smalltalk, views classes as object themselves, which still have an existence at run-time.”

B. Meyer Object-Oriented Software Construction

Some Class Properties

- Abstract: a class cannot have any instance
- Set: a class that knows all its instances
- DynamicIVs: Lazy allocation of instance structure
- LazyAccess: only fetch the value if needed
- AutomaticAccessor: a class that defines automatically its accessors
- Released/Final: Class cannot be changed and subclassed
- Limited/Singleton: a class can only have a certain number of instances
- IndexedIVs: Instances have indexed instance variables
- InterfaceImplementor: class must implement some interfaces
- MultipleInheritance: a class can have multiple superclasses
- Trace: Logs attribute accesses, allocation frequencies
- ExternalIVs: Instance variables stored into database

Understanding instantiation

- What is the relationship between an instance and its class?
- a class and its metaclass?
- a metaclass and its metametaclass?
- What is the difference between described-by and instances of?

At the Method Level

- Trace: Logs method calls
- PrePostConditions: methods with pre/post conditions
- MessageCounting: Counts the number of times a method is called
- BreakPoint: some methods are not run
- FinalMethods: Methods that cannot be specialized
Metaclass Responsibilities

- Classes as objects
- A class can be defined as a subclass of one or many other classes.
- A class is an object therefore instance of another class its metaclass that is an object too instance of a metaclass that is an object too instance of another metaclass.....
- To stop this potential infinite recursion
- Class is the initial class and metaclass
- Class is instance of itself and all other metaclasses are instances of Class

Unifying Class/Instance

- Every object is instance of a class
- A class is an object instance of a metaclass (P4)
- But all the objects are not classes
- Only one kind of objects without distinction between classes and final instances.
- Sole difference is the ability to respond to the creation message: new. Only a class knows how to deal with it. A metaclass is only a class that generates classes.

Roadmap

- Classes as objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- Class Structure
- Message Passing
- Object allocation & Initialization
- Class creation
- Inheritance Semantics
- Bootstrapping

ObjVlisp Postulates (I)

- P1: object = <data, behavior>
- P3: Every object belongs to a class that specifies its data (slots or instance variables) and its behavior. Objects are created dynamically from their class.
- P4: Following P3, a class is also an object therefore instance of another class its metaclass (that describes the behavior of a class).

ObjVlisp Postulates (II)

- Object
- Node
- accept:
- name
- send:
- Workstation

ObjVlisp 2nd Postulate

- P2: Message passing is the only means to activate an object

ObjVlisp 5th Postulate

- FS: A class can be defined as a subclass of one or many other classes.

Infinite Recursion

- A class is an object therefore instance of another class its metaclass that is an object too instance of a metametaclass that is an object too instance of another metametaclass.....

Why ObjVlisp?

- Minimal (only two classes)
- ObjVlisp self-described: definition of Object and Class
- Unifed: Only one kind of object: a class is an object and a metaclass is a class that creates classes
- Simple: can be implemented with less than 300 lines of Scheme or 30 Smalltalk methods.
- Equivalent of Clossett (Art of MOP example)
Metaclass

- Every object is instance of a class
- A class is an instance of a metaclass (P4)
- But all the objects are not classes
- Only one kind of objects without distinction between classes and final instances.
- Sole difference is the ability to respond to the creation message: new. Only a class knows how to deal with it.
- A metaclass is only a class that generates classes

• A class possesses the instance variable class inherited from Object that refers to its class (the metaclass that creates it).
• Classes as objects
  • A class has instance variables defined by a class
  • an ordered sequence of instance variables defined by a class
  - shared by all instances
  - values specific to each instance
  - In particular, every object possesses an instance variable class (inherited from Object) that points to its class.

Instance Structure

- Classes as objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
  • Class Structure
  • Message Passing
  • Object allocation & Initialization
  • Class creation
  • Inheritance Semantics
  • Bootstrapping

Methods

- Let’s use a Smalltalk block
  - name -> [:objself | objself unary: #name ]
- no direct access to instance variables

RoadMap

• Classes as objects
• ObjVlisp in 5 postulates
• Instance Structure and Behavior
  • Class Structure
  • Message Passing
  • Object allocation & Initialization
  • Class creation
  • Inheritance Semantics
  • Bootstrapping

RoadMap

- Classes as objects
• ObjVlisp in 5 postulates
• Instance Structure and Behavior
  • Class Structure
  • Message Passing
  • Object allocation & Initialization
  • Class creation
  • Inheritance Semantics
  • Bootstrapping

Instance Behavior

- Methods
  • belongs to a class
  • defines the behavior of all the instances of the class
  - is stored into a dictionary that associates a key (the method selector) and the method body
  • To unify instances and classes, the method dictionary of a class is the value of the instance variable methodDict defined on the metaclass Class.

Class as an Object

- A class possesses the instance variable class inherited from Object that refers to its class (the metaclass that creates it).
- Class value: an identifier of the class of the instance
- As an instance factory the metaclass Class possesses 4 instance variables that describe a class:
  - name: the class name
  - superclass: its superclass (we limit to single inheritance)
  - i-v: the list of its instance variables
  - methodDict: a method dictionary

Methods

- Let’s use a Smalltalk block
  - name -> [:objself | objself unary: #name ]
- no direct access to instance variables

Class Node as Object

- A class possesses the instance variable class inherited from Object that refers to its class (the metaclass that creates it).
- Class value: an identifier of the class of the instance
- As an instance factory the metaclass Class possesses 4 instance variables that describe a class:
  - name: the class name
  - superclass: its superclass (we limit to single inheritance)
  - i-v: the list of its instance variables
  - methodDict: a method dictionary

Class Node as Object

- A class possesses the instance variable class inherited from Object that refers to its class (the metaclass that creates it).
- Classes as objects
  • A class has instance variables defined by a class
  • an ordered sequence of instance variables defined by a class
  - shared by all instances
  - values specific to each instance
  - In particular, every object possesses an instance variable class (inherited from Object) that points to its class.

Class point as Object

- A class possesses the instance variable class inherited from Object that refers to its class (the metaclass that creates it).
- Classes as objects
  • A class has instance variables defined by a class
  • an ordered sequence of instance variables defined by a class
  - shared by all instances
  - values specific to each instance
  - In particular, every object possesses an instance variable class (inherited from Object) that points to its class.
Classes as objects
ObjVlisp in 5 postulates
Instance Structure and Behavior
Class Structure
Message Passing
Object allocation & Initialization
Class creation
Inheritance Semantics
Bootstrapping

RoadMap

Initial metaclass
Defines the behavior of all the metaclasses
Defines the behavior of all the classes

Message Passing

P2: Message passing is the only means to activate an object
P3: Every object belongs to a class that specifies its data and its behavior

Object Creation

Creation of instances of the class Point
- [Point new x 24 y 6]
- [Point new]
- [Point new y 10 y 15]
Creation of the class Point instance of Class
- [Class new
  name Point
  :super Object
  :v (x y)
  :methods (x ...display ...)
]

Object Creation: new

Object Creation = initialisation O allocation
Creating an instance is the composition of two actions:
memory allocation: allocate method
object initialisation: initialize method
(new aClass args) = (initialization (allocation aClass) args)
([aClass new] args) = ([aClass allocate] initialize args)
new creates an object: class or final instances
new is a class method
Object Allocation
- Object allocation should return:
  - Object with empty instance variables
  - Object with an identifier to its class
- Done by the method allocate defined on the metaclass
- allocate method is a class method

Allocation Examples
- [Point allocate] -> [Point nil nil] for x and y
- [Workstation allocate] -> [Workstation nil nil] for 'name' and 'nextNode'
- [Class allocate] -> [Class nil nil nil....]

Object Initialization
- Initialization allows one to specify the value of the instance variables by means of keywords (:x, :y) associated with the instances variables
- [Point new :y 6 :x 24] -> [Point nil nil] initialize (:y 6 :x 24)]
  - assign the values to the instance variables of the created object.

Metaclass Role
- Lookup method in the class of the receiver then we apply it to the receiver.

RoadMap
- Classes as objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- Class Structure
- Message Passing
- Object allocation & Initialization
- Class creation
- Inheritance Semantics
- Bootstrapping

Object Allocation
- Object with empty instance variables
- Object with an identifier to its class
- Done by the method allocate defined on the metaclass
- allocate method is a class method

RoadMap
- Classes as objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- Class Structure
- Message Passing
- Object allocation & Initialization
- Class creation
- Inheritance Semantics
- Bootstrapping

Instantiation Graph
- Class is the root of instantiation graph
- Object is a class that represents the minimal behavior of an object
- Object is a class so it is instance of Class

Object Initialization
- Initialization allows one to specify the value of the instance variables by means of keywords (:x, :y)
- Allocate method is a class method
- Object allocation should return:
  - Object with empty instance variables
  - Object with an identifier to its class
- Done by the method allocate defined on the metaclass
- allocate method is a class method

Class Creation
- Look in the class of the receiver

Object Initialization
- Object allocation should return:
  - Object with empty instance variables
  - Object with an identifier to its class
- Done by the method allocate defined on the metaclass
- allocate method is a class method

Object Allocation
- Object with empty instance variables
- Object with an identifier to its class
- Done by the method allocate defined on the metaclass
- allocate method is a class method
Two kinds of inheritance

- Static for the state
  - subclasses get superclass state
  - At compilation time
- Dynamic for behavior
  - inheritance tree walked at run-time

Instance Variable Inheritance

- Static for the instances variables
- Done once at the class creation
- When C is created, its instances variables are the union of the instance variables of its superclass with the instance variables defined in C.
- final-instance-variables (C) = Union (Union ( iv (super C)), local-instance-variables(C))

Method Inheritance

- Walks through the inheritance graph between classes using the super instance variable
- lookup (selector class receiver):
  - if the method is found then return it
  - else if receiver class = Object then [receiver error selector]
  - else we lookup in the superclass of the class
- the error method can be specialized to handle the error.

Inheritance Graph

- Object is the root of the hierarchy:
  - a Workstation is an object (should at least understand the minimal behavior), so Workstation inherits from Object
  - a class is an object so Class inherits from Object
  - In particular, class instance variable is inherited from Object class.

Inheritance Graph

- Object
- Class
- Workstation
- Node
- Point

Instance Variable Inheritance

- Object is the root of the hierarchy.
- Class is a Workstation.
- Workstation is an object.
- Class inherits from Object.
- In particular, class instance variable is inherited from Object class.

Method Inheritance

- Walks through the inheritance graph between classes using the super instance variable.
- lookup (selector class receiver):
  - if the method is found then return it
  - else if receiver class == Object then [receiver error selector]
  - else we lookup in the superclass of the class
- the error method can be specialized to handle the error.

Semantics of super

- As self, super is a pseudo-variable that refers to the receiver of the message.
- Used to invoke overridden methods.

Dynamic vs. Static

- self is dynamic:
  - Using self the lookup of the method begins in the class of the receiver.
  - Bound at execution-time
- super is static:
  - Using super the lookup of the method begins in the superclass of the class of the method containing the super expression (not in the superclass of the receiver class).
  - Bound at compile-time
super is NOT the receiver class superclass

- Let us suppose the WRONG hypothesis: “The semantics of super is to start the lookup of a method in the superclass of the receiver class”
- accept: is defined in DuplexWorkstation
- accept: is looked up in the class DuplexWorkstation
- accept: is not defined in DuplexWorkstation, so the lookup continues in Workstation

Yes... Why?

accept is defined in Workstation
lookup stops
method accept: is executed
Workstation>>accept: does a super send
Our hypothesis: start the lookup of the class of the receiver
=> superclass of class of a ColoredWorkstation is ... Workstation
Therefore we look in workstation again!!!

Minimal Shared Behavior

- The class Object represents the common behavior shared by all the objects:
  - classes
  - final instances.
- every object knows its class: instance variable class
- methods:
  - initialize (instance variable initialization)
  - error, class, metaclass?, class?
- Meta operations:
  - iv-set, iv-ref

RoadMap

- Classes of objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- Class Structure
- Message Passing
- Object allocation & Initialization
- Class creation
- Inheritance Semantics
- Some points
- Bootstrapping

About the 6th Postulate

- The ObjVlisp 6th postulate is:
  - class variable of anObject = instance variable of anObject's class
- So class variables are shared by all the instances of a class.

Why the 6th is wrong!

- Semantically class variables are not instance variables of object class!
- Instance variable of metaclass should represent class information not instance information shared at the meta-level.
- Metaclass information should represent classes not domain objects

Class initialization

- initialize is defined on both classes Class and Object:
- on Object: values are extracted from initarg list and assigned to the allocated instance
  - [m(name Point nil nil) initialize ([y 6 x 24])]
- => #(Point 6 24)
- Initialize is lookup in class of #(Point nil nil) : Point
- Then in its superclass: Object
Solution
A class possesses an instance variable that stores structure that represents instance shared-variable and their values.

Recap: Object class
- Defines the behavior shared by all the objects of the system
- Instance of Class
- Root of the inheritance tree: all the classes inherit directly or indirectly from Object
  - Its instance variable: class
  - Its methods:
    - initialize (initialisation les variables d’instance), error, class, metaclass?, class?, iv-set, iv-ref

Recap: Class class
- Initial metaclass
- Reflective: its instance variable values describe instance variables of any classes in the system (itself too)
- Defines the behavior of all the classes
- Inherits from Object class
- Root of the instantiation graph
- Instance variables: name, super, iv, methodDict
- Some Methods
  - new, allocate, initialize (instance variable inheritance, keywords, method compilation)
    - class?, subclass-of?

RoadMap
- Classes as objects
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- Class Structure
- Message Passing
- Object allocation & Initialization
- Class creation
- Inheritance Semantics
- Some points
- Recap
- Bootstrapping

Bootstrapping
- Mandatory to have Class instance of itself
- Be lazy: Use as much as possible of the system to define itself
- Idea: Cheat the system so that it believes that Class already exists as instance of itself and inheriting from Object, then create Object and Class as normal classes

Three Steps Bootstrap
- Manual creation of the instance that represents the class Class with
  - inheritance simulation (class instance variable from Object class)
  - only the necessary methods for the creation of the classes (new and initialize)
  - Creation of the class
    - Object [Class new'name 'Object'...]
    - definition of all the method of Object
  - Redefinition of Class
    - [Class new'name 'Class' super Object'...]
    - definition of all the methods of Class

Examples
- Metaclasses!
- ObjVlisp in 5 postulates
- Instance Structure and Behavior
- ...
- ...
- Examples

Abstract Classes
- The rule to define a new metaclass is to make it inherit from a previous one
- Prob: Abstract classes should not create instances
- Sol: Redefine the new method
Metaclass Definition

- [Class new :name Abstract :super Class :methods (new (lambda (self initargs) (self error "Cannot create instance of class %s" self name)))]
- Abstract is a class: It is instance of Class
- Abstract define class behavior: It inherits from Class

Metaclass Use

- [Abstract new :name Node :super Object ....]
- [Node new] -> Cannot create instance of class Node
- [Abstract new :name Abstract-Stack :super Object ....]

Method Lookup

S.Ducasse

References

- [Cointe'87] P. Cointe: “Metaclasses are First Class: the ObjVlisp Model”, OOPSLA'87.

Summary

Classes are objects too
Instantiation = initialize(allocate())
Class is the instantiation root
Object is the inheritance root
One single method lookup for classes and instances first go to the class then follow inheritance chain
super and self are referring to the message receiver but super changes the method lookup

License: CC-Attribution-ShareAlike 2.0
http://creativecommons.org/licenses/by-sa/2.0/