CLOS & MOP

Common Lisp Object System & Metaobject Protocol

Жикін Юрій
whythat@protonmail.com

18 травня, 2020

CLOS: Common Lisp Object System

- При підготовці стандарту ANSI Common Lisp у 1980-х запропонована як альтернатива більш раннім об'єктним системам *MIT Flavors* та *CommonLoops* в старіших мовах сімейства Lisp.
- Вся об'єктна система складається з трьох основних макросів (`defclass`, `defgeneric` та `defmethod`) та декількох функцій.

CLOS: термінологія

- **Загальна функція** (англ. *generic function*) - функція, реалізація якої обирається динамічно залежно від типів її параметрів (так званий *динамічний поліморфізм*).
- **Метод** (англ. *method*) - конкретна реалізація *загальної функції* для певної комбінації типів її параметрів.
- **Діючий метод** (англ. *effective method*) - метод, обраний під час виконання програми залежно від фактичних аргументів *загальної функції*.
- *Загальну функцію* можна розглядати як множину методів та правило вибору *діючого методу*.

CLOS: класи та об'єкти

- Класи оголошуються за допомогою форми `defclass`:

```
(defclass shape () ())

(defclass square (shape)
  ((side :accessor square-side
         :initarg :side)))

(defclass circle (shape)
  ((radius :accessor circle-radius
           :initarg :radius)))
```

- Створити об'єкт класу можна за допомогою функції `make-instance`:

```
(make-instance 'circle :radius 4)
```

CLOS: загальні функції та методи

- Загальні функції оголошуються за допомогою форми `defgeneric`:

```
(defgeneric perimeter (shape))
```

- Методи загальних функцій додаються формою `defmethod`:

```
(defmethod perimeter ((shape square))
  (* 4 (square-side shape)))

(defmethod perimeter ((shape circle))
  (* 2 pi (circle-radius shape)))
```

- Аналогічний приклад у мові Python:

```
class Square:
    def perimeter(self):
        return 4*self.side
```

CLOS: інтерфейси та мультиметоди

- Методи **не** є компонентами класів, тому інтерфейси в CLOS оголошуються як набір загальних функцій:

```
;; Graph interface.
(defgeneric node-payload (graph node))
(defgeneric previous-nodes (graph node))
(defgeneric next-nodes (graph node))
```

- Методи в CLOS можуть бути спеціалізовані відносно кількох параметрів – такий тип методів називають **мультиметодами**:

```
(defgeneric link-nodes (n1 n2))

(defmethod link-nodes ((n1 simple-node) (n2 simple-node))
  (add--line n1 n2))

(defmethod link-nodes ((n1 simple-node) (n2 comment-node))
  (add-dashed-line n1 n2))
```

CLOS: допоміжні методи

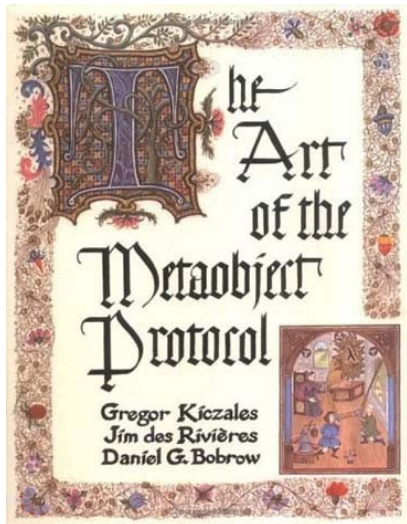
- CLOS дозволяє оголошувати методи, що адаптують існуючі методи, виконуючись перед, після чи “навколо” основного метода (модифікатори `:before`, `:after` та `:around`, відповідно):

```
(defmethod make-instance ((class class) &rest args)
   ...)

;; First find the class object and then call
;; the method above with it.
(defmethod make-instance :around ((name symbol) &rest args)
  (funcall #'call-next-method (find-class c) args))
```

MOP: Metaobject Protocol

- Система CLOS стала відома поза спільнотою користувачів Common Lisp завдяки книзі “Мистецтво метаоб’єктного протоколу” (Г. Кічалес, Дж. де Рів’єр та Д. Бобров), яку ідеолог об’єктно-орієнтованого програмування Алан Кей назвав “найкращою книгою за останні 10 років”.

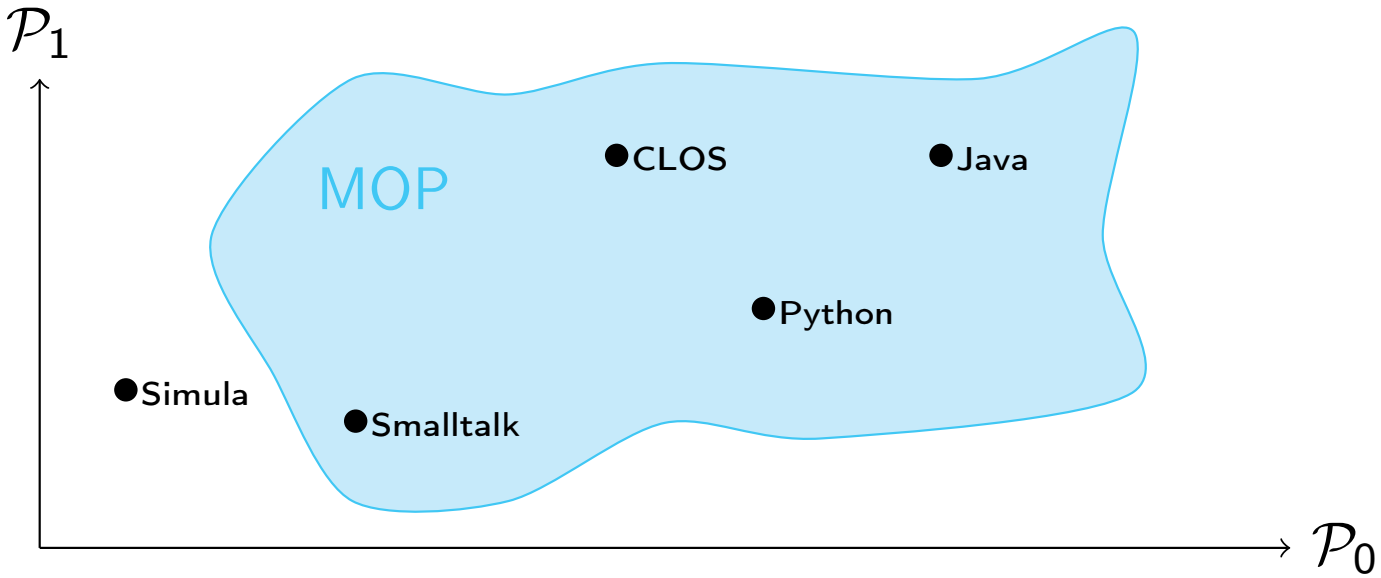


MOP: для чого?

- Принцип відкритості/закритості Бертрана Мейєра: *“об’єктні системи повинні бути відкритими для розширення, але закритими для модифікації”.*
- Один з основних принципів дизайну мови Common Lisp: *“користувач мови повинен мати змогу робити все, що може робити розробник мови”.*
- MOP задовільняє обидва принципи, подаючи саму об’єктну систему CLOS як набір CLOS-об’єктів.
- MOP доповнює макро-систему Common Lisp:
  - макро-система дозволяють користувачу створювати нові синтаксичні конструкції;
  - метаоб’єктний протокол дозволяє користувачу створювати нові об’єктні системи.

MOP: простір об'єктних систем

- MOP можна розглядати як область у просторі об'єктних систем, де самі об'єктні системи розглядаються як точки:



MOP: як це працює?

- Оголошення класів та загальних функцій зберігаються у глобальних реєстрах у вигляді об'єктів `class` та `generic-function` відповідно.
- Оголошення методів зберігаються у вигляді об'єктів типу `method` об'єктах `generic-functions`.
- Кожен новий клас додається у списки предків/нащадків відповідних класів. Ці списки фіксуються у якийсь момент перед створенням першого екземпляру класу.
- На основі списків пріоритету предків та списків методів для кожної загальної функції *будується* **функція-дискримінант**, яка реалізує вибір діючого методу залежно від аргументів.
- Під час виклику загальної функції викликається *функція-дискримінант*, яка обирає та викликає відповідний діючий метод.

MOP: об'єктно-орієнтований протокол

- В більшості реалізацій Common Lisp всі компоненти CLOS реалізовані як набір класів (метакласів), загальних функцій та методів CLOS.
- Спадкуючи нові метакласи від стандартних MOP-класів та перевантажуючи методи MOP, можна створювати нові об'єктні системи, що краще підходять для відповідної предметної області.
- Практичний приклад - оптимізація об'єктної системи.
- Приклад з історії - деякі legacy-бази коду, що використовували старіші мови Lisp занадто сильно залежали від деталей реалізації об'єктної системи Flavors, тому розробники портували їх на Common Lisp з адаптованою за допомогою MOP об'єктною системою, що поводилась як Flavors.

CLOS&MOP: на завершення

- CLOS - надзвичайно гнучка об'єктна система, що надає користувачу більше свободи та можливостей ніж більшість існуючих об'єктних систем в мейнстрімових мовах програмування.
- MOP розділяє функціональну та процедурну складові об'єктоно орієнтованої системи, відкриваючи можливості для оптимізацій, що не потребують змін в роботі компілятора.
- Окрім того, MOP має академічне застосування як платформа для експериментів з властивостями об'єктних систем.

Дякую за увагу!