Multiple Dispatch as Dispatch on Tuples

OOPSLA ’98

Gary T. Leavens

Iowa State University

Todd D. Millstein

University of Washington

Terms

Single dispatch (Smalltalk, C++, Java):

- Based on dynamic class of receiver only

  p1.equal(p2)

Multiple dispatch (CLOS, Dylan, Cecil):

- Based on dynamic class of many arguments

  equal(p1, p2)

Problem

Add multiple dispatch to single dispatch OO languages

1. Without changing:
   a. meaning, or
   b. typing
2. Keeping encapsulation properties

Idea

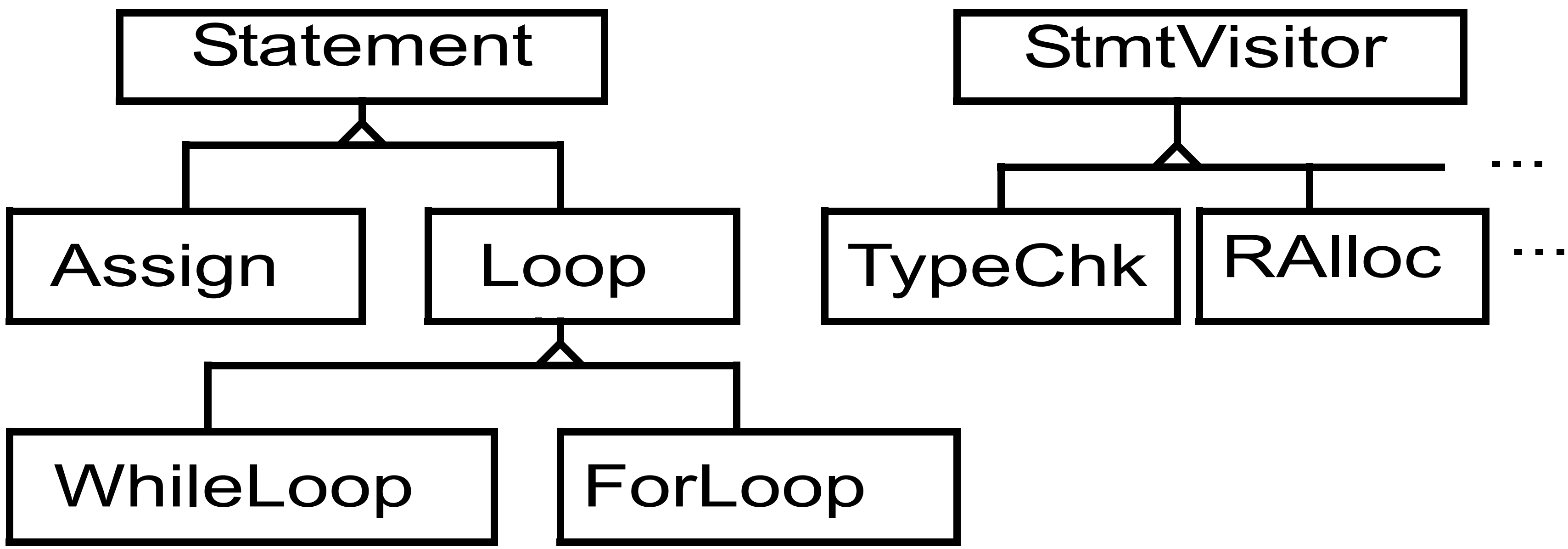
- Add tuples
- Send messages to tuples

  (p1,p2).equal()

Outline

- Why multimethods?
    - visitor pattern [GHJV95, BLR98]
    - binary methods
- Problems with workarounds
- Tuple mechanism
- Related work
- Conclusions

Visitor Pattern



- Code selected based on both Statement and Visitor

frlp.visit(typChk) ... typChk.visitForLoop(**self**)

(frlp,typChk).visit()

Binary Methods

```
class Point {
  fields (xval: int, yval: int)
  method x(): int { xval }
  method y(): int { yval }
  method distanceFrom(l: Line) : int { ... }


}

class ColorPoint inherits Point {
  fields (colorval: Color)
  method color(): Color { colorval }




}
```

```
method equal(p: Point): bool
  { xval = p.x() and yval = p.y() }
```

```
method equal(p: ColorPoint): bool -- type error
  { xval = p.x() and yval = p.y()
    and colorval.equal(p.color())
  }
```

Binary Methods Break Subtyping

```
class Break {
  fields ()
  method breakit(p: Point): bool
    { p.equal(new Point(3, 4)) }
}
```

Consider the message sends:

```
new Break().breakit(new Point(3, 5))

new Break().breakit(new ColorPoint(3, 5, red))
```

⇒ Subtype's methods cannot require more specific arguments

Outline

- Why multimethods?
- Problems with workarounds
  - instanceOf
  - double dispatch [Ing86]
  - overloading
  - product classes [BCC+95]
- Tuple mechanism
- Related work
- Conclusions

instanceOf

```
class Point {
   fields (xval: int, yval: int)
   method x(): int { xval }
   method y(): int { yval }
   method distanceFrom(l: Line) : int { … }
   method equal(p: Point): bool
      { xval = p.x() and yval = p.y() }
}

class ColorPoint inherits Point {
   fields (colorval: Color)
   method color(): Color { colorval }
   method equal(p: Point): bool
      { xval = p.x() and yval = p.y()
        and (not (p instanceOf ColorPoint)
                  or colorval.equal((ColorPoint)p).color()))
      }

}
```

Double Dispatch

```
class Point { ...
  method equal(p: Point): bool
    { p.equalsPoint(self) }
  method equalsPoint(p:Point): bool
    { xval = p.x() and yval = p.y() }
  method equalsCP(p:ColorPoint): bool
    { self.equalsPoint(p) }
}

class ColorPoint inherits Point { ...
  method equal(p:Point): bool
    { p.equalsCP(self) }
  method equalsCP(p:ColorPoint): bool
    { self.equalsPoint(p)
      and colorval.equal(p.color()) }
}
```

Problems with instanceOf and Double Dispatch

- Programmer codes the search
  $\Rightarrow$ complex code
- Need to modify existing code when adding classes
- Exponential number of methods

Overloading

```
class Point {
   fields (xval: int, yval: int)
   method x(): int { xval }
   method y(): int { yval }
   method distanceFrom(l: Line) : int { … }
   method equal_Point(p: Point): bool
      { xval = p.x() and yval = p.y() }
}

class ColorPoint inherits Point {
   fields (colorval: Color)
   method color(): Color { colorval }
   method equal_ColorPoint(p: ColorPoint): bool
      { xval = p.x() and yval = p.y()
        and colorval.equal_Color(p.color())
      }
}
```

Product Classes

```
class TwoPoints {
   fields(p1: Point, p2: Point)
   method equal(): bool
      { p1.x() = p2.x() and p1.y() = p2.y() }
}

class TwoColorPoints {
   fields(cp1: ColorPoint, cp2: ColorPoint)
   method equal(): bool
      { cp1.x() = cp2.x() and cp1.y() = cp2.y()
        and new TwoColors(cp1.color(),
                                   cp2.color()).equal()
      }
}

new TwoPoints(new ColorPoint(3,4,red),
              new ColorPoint(3,4,blue)).equal()
```

Evaluation of Product Classes

Problems:

- Loses dynamic dispatch
- No privileged access to objects

Advantage:

- Move binary methods out of normal classes
    - classes unchanged when add subclasses
    - subclasses are subtypes

Outline

- Why multimethods?
- Problems with workarounds
- Tuple mechanism
  - idea
  - example
  - modularity
- Related work
- Conclusions

Idea

Tuples + Dynamic Dispatch
= Multiple Dispatch

- Tuples:

  ()

  (myP1, myP2)

  (myDoc, myPrinter, 3)

- Dynamic dispatch on tuples:

  (myP1, myP2).equal()

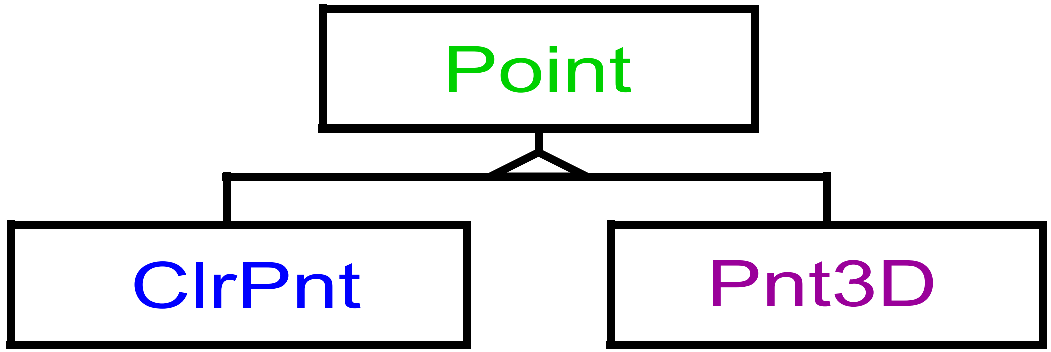
Example

```
tuple class (p1: Point, p2: Point) {
   method equal(): bool
      { p1.x() = p2.x() and p1.y() = p2.y() }
}

tuple class (cp1: ColorPoint, cp2: ColorPoint) {
   method equal(): bool
      { cp1.x() = cp2.x() and cp1.y() = cp2.y()
        and (cp1.color(), cp2.color()).equal()
      }
}


(new ColorPoint(3,4,red),
 new ColorPoint(3,4,blue)).equal()

(new Point(3,4),
 new ColorPoint(3,4,blue)).equal()
```

Most-Specific Applicable Method

```
tuple class (p1: Point, p2: Point) {
  method below(): bool { p1.y() < p2.y() }
}

tuple class (p: Point, cp: ColorPoint) {
  method above(): bool { p.y() > cp.y() }
  method below(): bool { p.y() < cp.y() }
}

tuple class (cp: ColorPoint, p: Point) {
  method above(): bool { cp.y() > p.y() }
  method below(): bool { cp.y() < p.y() }
}

(myPnt1, myClrPnt1).below()

(myClrPnt1, myClrPnt2).below() -- ambiguous

(myPnt1, myPnt2).above()        -- no method
```

Modularity Problem [Coo91]



Want to:

- Allow independent development from Point
- Type check separately

What Could Go Wrong?

2nd / 1st	Point	ClrPnt	Pnt3D
Point	(Point, Point)	(Point, ClrPnt)	(Point, Pnt3D)
ClrPnt	(ClrPnt, Point)	(ClrPnt, ClrPnt)	
Pnt3D	(Pnt3D, Point)	?	(Pnt3D, Pnt3D)

- Point and either ClrPnt or Pnt3D are ok
- Together they may have ambiguities

Solutions to Modularity Problem

- Extension Modules [CL95]
  - require unique most extending module
  - details not fully worked out
- Millstein & Chambers [MC98]
  - restrictions on where generic functions can be extended
  - soundness proof

Related Work

Generic Function Languages (CLOS, Dylan, Cecil, ...)

- Don’t say what arguments they dispatch on:

f(x,y,z) vs. (x,y).f(z)

- More uniform
- Can’t add multiple dispatch to existing languages
- Encapsulation is not class-based

Encapsulated Multimethods

```
class ColorPoint inherits Point {
   fields (colorval: Color)
   method color(): Color { colorval }
   method equal(p: Point): bool
      { xval = p.x() and yval = p.y() }
   method equal(p: ColorPoint): bool
      { xval = p.x() and yval = p.y()
        and colorval.equal(p.color()) }
}
```

Related Work

Encapsulated Multimethods [BCC+95, Cas97]

- May need to edit code when adding subclass
- Could have privileged access...
- ... but only to data of same class
- No modularity problems ...
- ... but need to duplicate methods

Parasitic Methods [BC97]

- Avoids code duplication by multimethod inheritance
- Textual ordering with inheritance is more complex

Conclusions

- Way to add multiple dispatch to single-dispatch languages
- Simple, elegant
- No effect on existing code
- No change to typing
- No change to information hiding properties

References

[BC97] John Boyland and Giuseppe Castagna. Parasitic methods: Implementation of multi-methods for Java. In *OOPSLA '97, 12th ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 66–76, 1997. Volume 32, number 10 of *ACM SIGPLAN Notices*.

[BCC+95] Kim Bruce, Luca Cardelli, Giuseppe Castagna, The Hopkins Object Group, Gary T. Leavens, and Benjamin Pierce. On binary methods. *Theory and Practice of Object Systems*, 1(3):221–242, 1995.

[BLR98] Gerald Baumgartner, Konstantin Läufer, and Vincent F. Russo. On the interaction of object-oriented design patterns and programming languages. Technical Report CSD-TR-96-020, Department of Computer Science, Purdue University, February 1998.

[Cas97] Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science. Birkhauser, Boston, 1997.

[CL95] Craig Chambers and Gary T. Leavens. Typechecking and modules for multi-methods. *TOPLAS*, 17(6):805–843, November 1995.

[Coo91] William R. Cook. Object-oriented programming versus abstract data types. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, New York, N.Y., 1991.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[Ing86] Daniel H. H. Ingalls. A simple technique for handling multiple polymorphism. *ACM SIGPLAN Notices*, 21(11):347–349, November 1986. OOPSLA '86 Conference Proceedings, Norman Meyrowitz (editor), September 1986, Portland, Oregon.

[MC98] Todd Millstein and Craig Chambers. Modular statically typed multimethods. Technical Report UW-CSE-98-07-01, University of Washington, July 1998.