Prototypes with Multiple Dispatch: An Expressive and Dynamic Object Model

Lee Salzman Jonathan Aldrich

Carnegie Mellon University

July 28, 2005

Outline

1. Benefits of prototypes and multiple dispatch
2. Challenges in combining prototypes and multiple dispatch
3. PMD: a new model of multiple dispatch
4. Slate: practical experience with PMD

Why Prototypes?

Objects represent themselves (without classes) by describing their own methods and inheritance relationships.

Benefits

- Simpler language kernel
- Metaprogramming
- Interactive and incremental development

Why Multiple Dispatch?

All arguments to a method invocation participate in dispatch, not just the first.

Benefits

- fewer restrictions on code factoring and reuse
- don't need to use simulations such as double dispatch or visitor pattern

Example

Integer + Integer
Float + Fraction
Complex + Float

Combining Prototypes and Multiple Dispatch

	Self [Ung 87] prototypes	Cecil [Cha 92] prototypes + multiple dispatch
dispatch	dictionaries containing methods	methods constrained to specific objects or objects inheriting them
cloning	copies all methods	may only copy fields and must inherit methods
method update	anywhere	may only define methods at top–level
inheritance	delegation	inclusion (fixed) and predicate dispatch (dynamic) [Cha 93]

Combining Prototypes and Multiple Dispatch: Generic Functions

- prior multiple dispatch approaches rely on generic functions [Bob 88] or similar mechanisms
- generic function groups together all methods with similar name and arity
- apply generic function to invoke a method
- generic function selects applicable methods by checking method constraints against arguments
- orders applicable methods by constraints to find the most specific one

Combining Prototypes and Multiple Dispatch: Generic Functions Won't Work

- dispatch information stored in external constraints – not encapsulated
- expensive to test and order all the constraints at the time of dispatch
- implementations generate dispatch tables or decision trees based on static inheritance relationships
- ideal for use with classes where inheritance relationships are fixed
- problematic for prototypes where inheritance may be unpredictable

PMD: A Paradigm Shift

Requirements

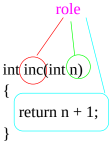
- must internalize dispatch information into objects
- all method arguments must decide the result of dispatch
- must be practical to evaluate inheritance at time of dispatch

Solution: roles

- Objects may play roles in a method corresponding to the method's parameters.
- A method represents an interaction in which all necessary roles have been fulfilled.
- Only objects know which roles they may fulfill.

PMD: Concepts

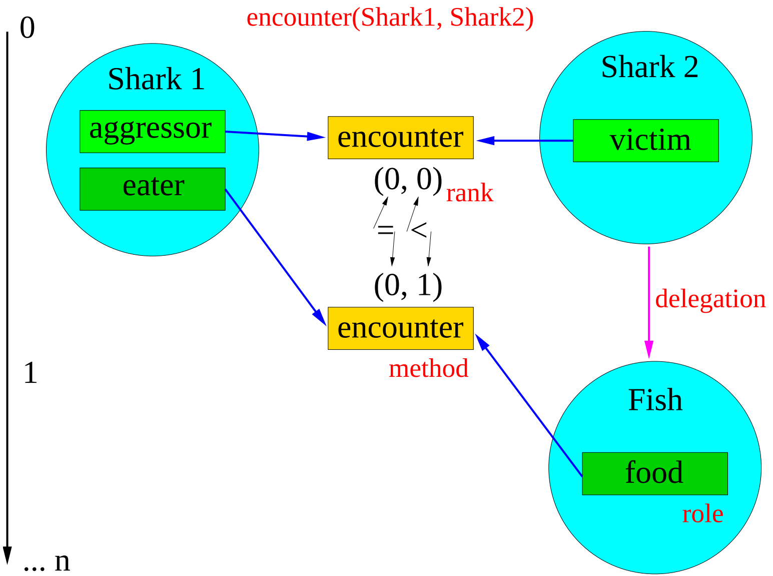
- A role identifies, for a method definition, a method name and parameter for which an object agrees to be used, as well as a method to be run should the role be satisfied.
- An object is a set of roles and delegation relationships, and roles may be inherited by delegation.
- A role is satifisfied if the name of the method to be invoked and the argument position where the role was found invocation match those described by the role.
- A method is applicable if there is a set of satisfied roles referring to it that cover all arguments to an invocation.
- Applicable methods are ranked according to the positions of their corresponding roles in the delegation hierarchy.



role

```
int inc(int n)
{
  return n + 1;
}
```

PMD: Example



0
encounter(Shark1, Shark2)
Shark 1
aggressor
eater
encounter
(0, 0)
rank
=
<
(0, 1)
encounter
method
Shark 2
victim
delegation
Fish
food
role
1
... n

PMD: The Calculus

Concepts

- roles
- object identity modeled through object store
- method update based on roles
- dynamic inheritance of roles through delegation
- multiple dispatch based on roles

Abstracts over:

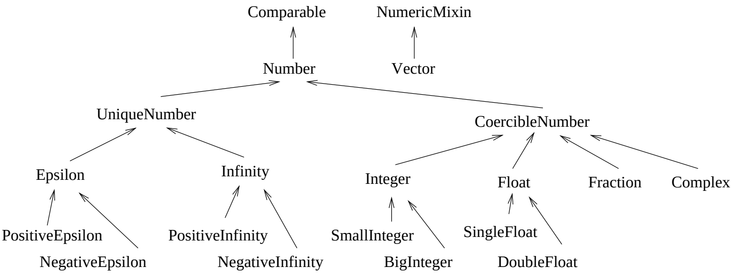
- inheritance order
- precedence of method parameters

Slate

- programming language based on PMD and strongly inspired by Self
- object model largely the same as Self, with provisions for roles
- incorporates many organizational concepts from Self without loss (namespaces and traits)

Slate: Further Experience

- multiple dispatch extensively used in libraries such as numerics, collections, streams and the compiler
- libraries designed to take advantage of multiple dispatch and benefited from it
- allowed for practical integration of large amounts of objects



Slate: Subjectiveness?

Us [Ung 96]

- perspective-receiver symmetry for subjectiveness
- dispatch on perspective and receiver
- perspectives dynamically composed with layers
- noted multiple dispatch allows for similar benefits

Potential uses

- security
- multi-user

PMD

- prototypes useful for creating and composing unique or shared perspectives
- multiple dispatch embeds subjectiveness with only a few changes

Summary

- PMD consistently combines prototypes and multiple dispatch.
- PMD provides a conceptual understanding of why they combine.
- PMD's roles internalize dispatch information.
- PMD allows for flexible objects with fewer restrictions.