Multiple Dispatching

Alex Tritthart

WS 12/13
Outline

1. Introduction

2. Dynamic Dispatch
   - Single Dispatch
   - Double Dispatch

3. Multiple Dispatch
   - Example

4. Evaluation
What is it all about?

Assume

```cpp
class Animal { shout() = 0 }
class Lion : Animal { shout() {"roar"} }

void shoutMethod(Animal &a) { a.shout() }
```

Call

```cpp
Lion lion();
shoutMethod(lion); // should output "roar"
```
What do we need?

We need

- no compiling time resolution, but **runtime function resolution**
- no function overloading, but **function overriding**

This is called:
Dynamic Dispatch
What do we need?

We need

- no compiling time resolution, but **runtime function resolution**
- no function overloading, but **function overriding**

This is called:

Dynamic Dispatch
Dynamic Dispatch

- Decision at runtime
- Allows instance behaviour
- Implemented through virtual tables

**Dynamic Dispatch is every Dispatch with:**

- More than one possibility
- Same method signature in super class and subclasses
- Choosing method depending on the **real** types of arguments

There exist single dispatch, double dispatch, triple dispatch, ...
Definition

"The Object-Oriented term for overriding, allowing different methods to be used depending on the type of a single object."
Single Dispatch

- Dynamic dispatch with a single type
- Used mostly in object oriented languages
- Simple to implement

Languages
Java, C#, JS, Python, Ruby can handle single dispatch
But how to realize?

Virtual Methods

Virtual Tables
But how to realize?

Virtual Methods

Virtual Tables
Virtual Methods

- Methods which behaviour can get overridden
- Methods where the vector address gets determined at runtime
- Requires: same class hierarchy and same signature

Example

class Animal : public Animal{
    virtual void shout() = 0;
}

Languages
- In C++, Object Pascal an explicit declaration is necessary
- In Java, Smalltalk, Python all methods are virtual
Virtual Function Table

- Each function gets an index
- Each object gets a hidden pointer to the table
- The compiler **only** generates the routing - so calls can be dissolved

Alternatives

- Binary tree dispatch
- String in a hash table
Virtual Function Table
Double Dispatch

Considers behavior between two objects with subclasses

Examples
- Adaptive collision
- Event handling systems
- Printing figures on different printers

Not straight supported by OO languages like java and C++
Print all figures on all printers

```java
void printAllEverywhere(Figure[] figures, Printer[] printers)
for (i : figures) {
    Figure figure = figures[i];
    for (j : printers) {
        Printer printer = printers[j];
        figure.printOn(printer);
    }
}
```

must work for any printer or figure!
The Problem with Double Dispatch

Assume

```java
Figure figure;  // default: circle−figure
SquareFigure squareFigure;
Printer printer;  // default: black−white−printer
ColorPrinter colorPrinter;
```

Call

```java
squareFigure.printOn(printer);  // square in black and white
squareFigure.printOn(colorPrinter);  // square in color
```
The Problem with Double Dispatch

Assume

\[ \text{Figure} \& \ \text{figureReference} = \text{squareFigure} ; \]

Call

\[ \text{figureReference}.\text{printOn}(\text{printer}) ; \quad // \text{square in black and white} \]
\[ \text{figureReference}.\text{printOn}(\text{colorPrinter}) ; \quad // \text{square in color} \]
The Problem with Double Dispatch

Assume

```java
Figure& figureReference = squareFigure;
Printer& printerReference = colorPrinter;
```

Call

```java
// note the type of the pointer and the type of the object.
figure.printOn(printerReference); // circle in color?
figureReference.printOn(printerReference); // square in color?
```
Multiple Dispatch

- Uses runtime types of more than one argument
- Generic function chooses the appropriate method
- The choice depends on the signature
- Multiple Dispatch is a kind of nested instance-of checks

Languages

- Common Lisp, Haskell, Dylan, Nice, Perl 6, C# 4.0
- With extensions: Python, Perl, Java, C#, .NET
Multiple Dispatch Pattern
Example in C# 4.0:

```csharp
void printOnImpl(Figure x, Printer y) {
    // print circle in black and white
}
void printOnImpl(SquareFigure x, Printer y) {
    // print square in black and white
}
...

void printOn(TopClass x, TopClass y) {
    dynamic a = x;
    dynamic b = y;
    printOnImpl(a, b);
}
```
Advantages

- Less code to write
- no argument preparation
- Avoiding asymmetry of single dispatch
- Generic function is relevant for several classes
- Extending a class becomes nearly trivial
- Languages with multiple dispatch use efficient algorithms to resolve calls
Disadvantages

- Single dispatch occurring most, already useable by most object oriented languages
- Median overhead of 13.7% to the maximum of 47% (study 1996)[4]
- There are not that many real cases where double or triple dispatch is used
- Different generic functions for one class
- **But**: Development tools help finding all relevant functions
When should I use multiple dispatch?
Whenever you are implementing equal methods which work with a number of different objects

Question:
Is there a workaround to get double dispatch in languages with using single dispatch?
Workaround: Visitor Pattern

Add method to printers:

```cpp
virtual void printOnImpl(Figure& figure)
    figure.printOn(*this);
```

Call

```cpp
Printer& printerReference = colorPrinter;
Figure& figureReference = squareFigure;
printerReference.printOn(figure);
printerReference.printOn(figureReference);
```
Evaluation

Quellenangabe

http://c2.com/cgi/wiki?MultipleDispatch

http://c2.com/cgi/wiki?DoubleDispatchExample


The multiple dispatch pattern in its detail


http://java.dzone.com/articles/multiple-dispatch-fix-some