Programming Languages (Langages Evolués)

Roel Wuyts

Introduction

Responsibles

- Roel Wuyts
  - roel.wuyts@ulb.ac.be
  - Office: N8.111 (temporarily: O8.214).
  - You can always come by for questions.
    - But: Always make an appointment first!!
- Assistent: Vandy Berten
  - vandy.berten@ulb.ac.be
- http://www.ulb.ac.be/di/rwuyts/INFO020_2003/

About this course

- It is based on some existing courses:
  - Nierstrasz: Programmiersprache
  - Ducasse: "OOP with Smalltalk".
  - Mens: Langages de Programmation
- ... and on my own courses.

What is this course about?

- Theory:
  - Programming Language Paradigms
    - OO, Functional, Logic
  - Programming Languages
    - Self, Smalltalk, Scheme, Prolog, ...
  - Evaluating paradigms and languages
    - in function of problem, elegance, efficiency, ...
  - Meta programming and reflection

What is this course about?

- Practice:
  - Apply theory in practice
  - Exercises in several concrete languages
    - Scheme (functional programming)
    - Self (object-oriented programming)
    - Prolog (logic programming)
  - Write in a "functional style" in an object-oriented language, etc.

Examination

- Exam at end of year.
  - Theoretical and practical part:
    - theoretical concepts
    - exercises

Bit of History

- Generations of Programming Languages:
  1. machine codes
  2. symbolic assemblers
  3. (machine independent) imperative languages (FORTRAN, Pascal, C ...)
  4. domain specific application generators (PowerBuilder)
  5. high-level languages (Prolog, Smalltalk, Lisp)
  6. ??

Safir-Worff Hypothesis

- Safir-Worff hypothesis for natural language:

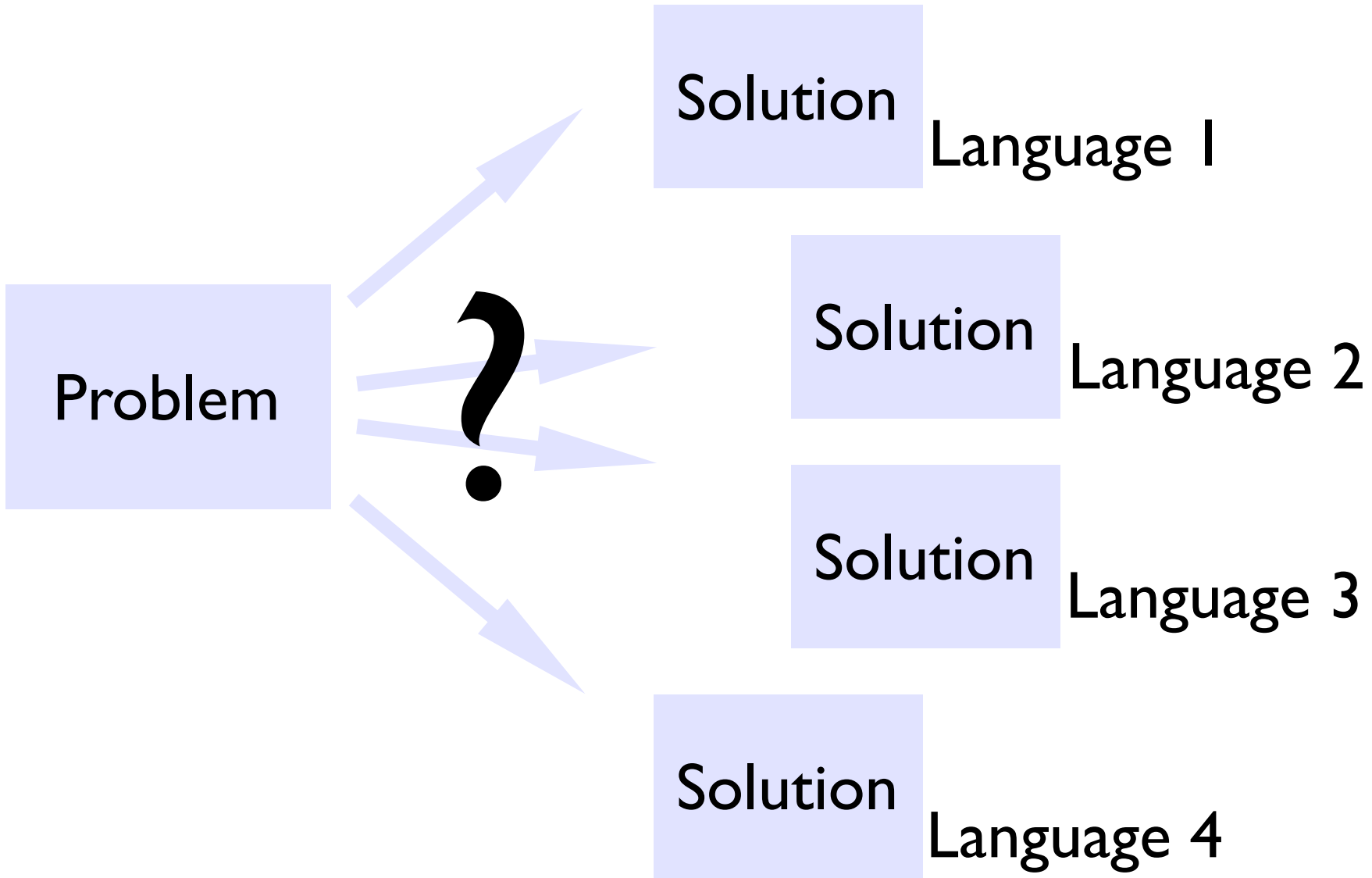
  Language helps us determine how we see and think about the world.

  - Language restricts the thought of people who use it
  - The limits of one's language become the limits of one's world.
  - Example: Eskimos have many names for snow so they think about snow differently.

Safir-Worff for Programming

- *Programming language determines how we see and think about the solution.*
- I don't believe that a good programmer is as efficient regardless of the language used!
  - Neither is the program...
- A programming language is a problem-solving tool
  - Choose one explicitly in function of your problem!

Programs as Solutions

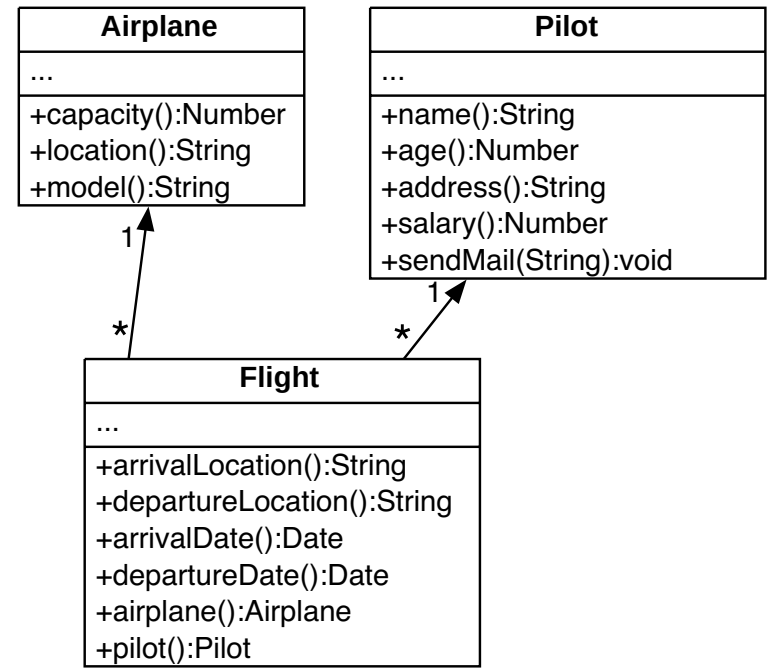


Motivating Example

- Problem:
  - Suppose that there is a "*datastore*" with information about flights, pilots and airplanes
  - Then print the names of all the pilots, ranked by salary in increasing order, in charge of a flight to Paris on an A320 airplane.
- Let's solve it in a functional, OO and logic way

OO (Java)

```
TreeSet pilots = new TreeSet(new Comparator()
{
   public int compare(Object o1, Object o2)
   {
      int salary1 = ((Pilot)o1).salary();
      int salary2 = ((Pilot)o2).salary();
      if (salary1 < salary2)
         return -1;
      else if (salary1 == salary2)
         return 0;
      else
         return 1;
   }
});
Iterator flightIterator = F.iterator();
while (flightIterator.hasNext())
{
   Flight currentFlight = (Flight)flightIterator.next();
   if (currentFlight.arrivalLocation().equals("PARIS")
   && currentFlight.airplane().model().equals("A320"))
   {
      pilots.add(currentFlight.pilot());
   }
}
Iterator pilotIterator = pilots.iterator();
```



OO (Smalltalk)

```
| parisA320Salaries |
Plane repository
   add: (Plane number: #sn023 type: 'A320' base: 'Brussels');
   add: (Plane number: #aif005 type: 'B747' base: 'Paris');
...
Pilot repository
   add: (Pilot name: #Marzi age: 41 salary: 36358);
...
Flight repository
   add: (Flight from: 'Brussels' to: 'Paris'
                  on: 031219 number: #sn023 pilotName: #Marzi);

parisA320Salaries := SortedCollection sortBlock: [:pilot1 :pilot2 |
                                    pilot1 salary < pilot2 salary].
Flight repository do: [:flight |
   ((flight isTo: 'Paris') & (flight isPlaneType: 'A320'))
         ifTrue: [parisA320Salaries add: flight pilot]
].
^parisA320Salaries
```

class

block

object

messages

FP (Scheme)

```
(let ((flights '((Brussels Paris 031219 sn023 Marzi)
                 (Londen Paris 031219 sn023 Jean)
                 (Paris Londen 031219 sn023 Roger)
                 (Paris Amsterdam 031220 sn02 Marzi)))
      (planes '((sn023 A320 Brussels)
                ('aif005 B747 Paris)
                ('sno2 A320 Brussels)))
      (pilots '((Marzi 41 36358)
                (Jean 29 29582)
                (Roger 54 41672))))

  (define (isParisA320Flight? flight)
    (let* ((planeAndPilot (cdddr flight))
           (plane (car planeAndPilot)))
      (and (eq? (cadr flight) 'Paris)
           (eq? (cadr (assoc plane planes)) 'A320))))

  (define (pilotSalary flight)
    (let* ((planeAndPilot (cdddr flight))
           (pilot (cadr planeAndPilot)))
      (list (caddr (assoc pilot pilots)) pilot)))

  (display (quicksort (map pilotSalary (filter isParisA320Flight? flights))
                      (lambda (x y) (< (car x) (car y))))) (newline)
  )
```

data definition

functions

LP (Prolog)

```
flight(brussels, paris, 031219, sn023, marzi).
flight(londen, paris, 031219, sn023, jean).
flight(paris, londen, 031219, sn023, roger).
flight(paris, amsterdam, 031221, sn02, marzi).

airplane(sn023, a320, brussels).
airplane(aif005, b747, paris).
airplane(sn02, a320, brussels).

pilot(marzi, 41, 36358).
pilot(jean, 29, 29582).
pilot(roger, 54, 41672).


parisA320Salaries(SortedPilots) :-
   findall(  -(Salary, Pilot),
             (  flight(_, paris, _, Plane, Pilot),
                airplane(Plane, a320, _),
                pilot(Pilot, _, Salary)),
             UnsortedPilotList),
   keysort(UnsortedPilotList, SortedPilots).
```

facts

query

Notes on these Examples

- Compare the Java Iterator with the Smalltalk do:, and the Java nested class with the Smalltalk block
  - Smalltalk uses *functional* blockclosures
- Compare the ways the data is expressed
- Compare the 'query' expressions
- Compare the usage of types

Language Comparison

- Criteria
  - Readability
  - Elegance
  - Efficiency
  - Expressivity
  - ...

Paradigms

- A paradigm is a model or pattern, an archetype, for solving problems. Inherent in a paradigm are basic assumptions about problems. Consequently a paradigm defines or structures how problems are approached.
- A paradigm provides a broad philosophy to be adopted during the development process. It describes the general approach to be used, rather than the specific tools and methods.

Programming Paradigms

- One classification:

Imperative	Program = Data + Control Focus lies on "How" C, Java, Smalltalk, functional programming with assignments
Declarative	Program = Data Focus lies on "What" SQL, Prolog, functional programming without assignments

Programming Paradigms

- Other classification (cross-cutting the first one):

Procedural	Procedures and control loops Basic, Pascal, Modula-2, ...
Functional	Functions and recursion Lisp, Scheme, ...
Object-Oriented	Objects and message sending Simula, Smalltalk, C++, Self, Java, JavaScript, ...
Logic	Clauses and unification and backtracking Prolog, ...

Other Paradigms

- Concurrent Programming
- Distributed Programming
- Constraint Programming
- Event-driven Programming
- Stream Programming
- ...

Course Overview

- Object-Oriented Programming
- Functional Programming
- Logic Programming
- Meta-Programming and Reflection
- Multiparadigm programming
- Selected Research Topics: Language symbiosis, Traits, ClassBoxes, Components

Wrap-up

- This course is about programming languages, and about using a programming language as a tool
- The importance of the Saffir-Worf hypothesis
- Programming paradigms and their differences

References

- Course URL: http://www.ulb.ac.be/di/rwuyts/INFO020_2003/
-
- The evolution of programming languages
  Peter Grogono, Course Notes, 1999 (on website)
- Concepts of Programming Languages, 6th edition
  Robert W. Sebesta, Addison Wesley, 2003
- Principles of Programming Languages, 3rd edition
  Bruce J. Maclennan, Oxford University Press, 1999
- Programming Languages: Principles and Paradigms
  Allen Tucker & Robert Noonan, McGraw Hill, 2002
- Concepts, Techniques, and Models of Computer Programming
  Peter Van Roy, Seif Haridi, MIT Press, 2003 (on website)