First Steps in AmbientTalk

Sequential programming

Elisa Gonzalez Boix
egonzale@vub.ac.be

Lab Sessions Schedule

W	Date	Exercise	Concepts
22	12/02/2012	First steps in Android - Simon	Android programming
23	19/02/2012	First steps in AmbientTalk	Sequential programming, Java symbiosis
24	26/02/2012	Internet Cafe	Concurrent programming, unit test
25	05/03/2012	Mobile Music Player	Distributed programming, Failure Handling
26	12/03/2012	weScribble on Android devices	Distributed programming, Java symbiosis
27	19/03/2012	Flikken in TOTAM	Tuple-based distributed programming
28	26/03/2012	wePoker on Android devices	Distributed programming, Java symbiosis
EASTER HOLIDAY			
31	16/04/2012	goShopping with REME-D	Reflective progr.,Distributed Debugging
32	23/04/2012	Omnireferences	Reflective progr., Intercession
...			
39	10/06/2012	Project delivery	report + code
40/1	17-30/06/2012	Project defenses	30-minute discussion with demo

---

Project

- Implement a distributed application in AmbientTalk.
- Evaluated mostly on good distributed design.
- What is important to remember?
  - Individual!
  - Quality and structure of the code!
  - Test cases and report!

Material

http://soft.vub.ac.be/amop/

- Language reference
- Tutorial
- Lab sessions material

http://code.google.com/p/ambienttalk

- AmbientTalk IDE for Eclipse (IdeAT)
- iat command line parameters

Variables, functions & tables

Definition

- As in Pico:

```
def x := 5

def square(x) { x * x }

def t[<size>] { <expression> }
```

Assignments

- Almost as in Pico:

```
x := 5

square := { |x| x * x }

t[1] := 5
```

Referencing

- As in Pico:

```
x

square(5)

t[1]
```

Functions

- Support for lambda's: closure literal

```
{|a,b| a + b };

def square := { |x| x * x };
square(2);
```

Functions

- Variable length arguments

```
def sum(@args){
  def total := 0;
  foreach:{ |el| total := total +el} in: args;
  total;
};
```

- Optional arguments

```
def incr (num, step := 1) { num + step };
incr(3);
incr(3,3);
```

'built-in' control structures

- Defined in the lexical root (top-level)

Check
the language
reference

```
if: (n < 1) then: { ... } else: { ... }

def if: cond then: cons else: alt {
  cond.ifTrue: cons ifFalse: alt
}

while: { i<10 } do: { ... }

foreach: table in: foo
```

Objects

Prototypical Objects

- Ex-nihilo creation:

```
def point := object: {
  def x := 0;
  def y := 0;
  def sumofsquares() { x*x + y*y };
}

point.x;
point.sumofsquares;
```

Cloning and Instantiation

- new = clone + init

```
def Point := object: {
  def x := 0;
  def y := 0;
  def init(anX, aY) {
    x := anX;
    y := aY;
  }
}

def anotherPoint := Point.new(2, 3);

Point.x >> 0
anotherPoint.x >>2
```

Object Extension

- Clones the parent + implicit delegation

```
def Point3D := extend: Point with:{
  def z := 0;
  def init(anX, aY, aZ) {
    super^init(anX, aY);
    z := aZ;
  }
}

def anotherP3D := Point3D.new(1,2,3);
```

Lexical Scope

- Objects have full access to enclosing environment of definition.

```
def makePoint(anX, aY) {
  object: {
    def x := anX;
    def y := aY;
    def sumofsquares() { x*x + y*y };
  }
}
```

Lexical Scope

- Nesting objects is allowed:

```
def point := object: {
  def x := 0;
  def y := 0;
  def sumofsquares() { x*x + y*y };
  def prettyprinter := object: {
    def print() { “(“+ x +”,”+ y +”)” }
  }
}
```

Object Scope

- Object = slots (= fields + methods)

  + lexical parent + dynamic parent

```
def o := object: {
  def x := 5;
  def getStatic() { x };
  def getDynamic() { self.x };
}

def o2 := extend: o with: {
  def x := 6;
}
```

o2.getStatic ??
o2.getDynamic ??

Native Data Types

- numbers, fractions, text, tables, booleans
- all objects: former ‘native functions’ are now ‘native methods’.
- Text:

```
          "AmbientTalk".explode();

   "a;b;c".split(";"); //["a","b","c"]
```

- Numbers:

```
6.to: 0 step: 2 do: { |i|
  system.println(i)
} // 6 4 2
```

Keyworded Messages

- Just a special type of selector

```
def util := object: {
  def map: fun onto: tbl {
    def i := 0;
    def copy[tbl.length] { fun(tbl[i:=i+1]) };
    copy;
  }
}


util.map: { |x| x*x } onto: [1,2,3]
```