Functional reactive Scala for 2D painting

Michel Ganguin

February 21, 2007

Overview

Introduction

- Functional reactive abstract values that changes over time by using signals.
- It is possible to bind a function to a signal which, is constantly applied to the value.
- Such an abstraction becomes very interesting in GUI programming, which is almost based on events (user actions) and on time (animations).

Signals

- A signal has an initial value and its current value is always accessible.
- Bind a function to it creates a new signal depending on the initial one. The function is registered but not evaluated.
- It is evaluated every time when the value is accessed.

A Signal example

```
val fstText = IOSignal("Hello")
val sndText = IOSignal("world!")

val both = for (val fst <- fstText;
                val snd <- sndText)
             yield fst + "␣" + snd

for (val b <- both) Console.println(b)
```

What happens with this code?

A Signal example (cont)

Just another simple transformation

```
val BOTH = for (val b <- both)
                 yield b.toUpperCase()

for (val b <- BOTH) Console.println(b)
```

And when the carried value changes?

```
fstText.current="Hi"

sndText.current="earth!"
```

Events

- An event has neither an initial value nor an accessible current value.
- As for Signals, function can be bound.
- Events combination is not possible (would never happen).
- Or | can be used.
- It can be transformed into a signal with holding the value of the last occurence.

An Event example

```
val click = new EventSource[MouseEvent]

def mouseClicked(e: MouseEvent): Unit =
    click fire e

val clickpos = for (val c <- click)
    yield c.getPos()

for (val cp <- clickpos)
    Console.println(cp)

on(click)(Console.println("clicked"))
```

Java 2D

- Some paintable elements are not objects (ex: String)
- Some properties are global and not in objects (ex: color)
- Repainting has to be managed manually

Fresca 2D

- All paintable elements are objects (Shape)
- Properties are mixed in Shape and do not affect other shapes
- There is a special Shape for global properties
- Repainting is managed automatically
- Shape is a signal

Example

```
new PaintBoard with mouse {
  val text =
    Constant("I␣follow␣the␣mouse")
  val x = mouse.pos.map(p => p.getX())
  val y = mouse.pos.map(p => p.getY())

  val str = new String2D(text, x, y)

  val shapes = str :: Nil
}
```

with properties

```
val str = new String2D(text, x, y)
            with Color with Transform {
  val color =
    mouse.pressed.map(p => if (p) RED
                           else BLUE)

  val matrix = Transform.rotate(
        Time.seconds.map(s =>
            (2. * Pi) / 60 * (s%60)),
        x, y)
}
```

GlobalShape

- GlobalShape is an exception that has global effect
- with Color: sets the background color
- with Transform: perform a global transformation (applied to every following shapes)

A new Shape

- If it is a java.awt.Shape, simply create a new class that extends AWTShape
- Otherwise,
    - Create a case class that extends Shape
    - Add a case to the match in method frpaint of trait Paint
    - Specify the behaviour for each applicable properties (unapply them if necessary)
    - Write the procedure to draw the Shape
    - If wanted, add a case to the match of method setShapes of trait Paint, to specify how to call repaint (otherwise full scene will be repainted on changes)

A new property

- Create a trait with self type Shape (or a subclass of Shape), that contains an absract field with the new property signal.
- Update shape signal on every property signal change.
- For all concerned shapes, specify what to do if property is mixed in (in method frpaint in trait Paint)

Conclusion

- Functional reactive programming offers a powerful way to draw animations, to handle user interactions.
- In my opinion, code is easier to write, easier to read and shorter.

The End - Questions?