ORC

Almost an introduction

Paolo Baldan

Linguaggi e Modelli per il Global Computing
AA 2013/2014

Aims

- A concurrent language should
  - Describe entities and their interactions
  - Allow birth and death of entities.
  - Allow programming of novel interactions.
  - Support hierarchical structure.
  - Describe passage of time.

Idea

- Internet scripting language: integrate and coordinate existing services (ORChestrate)
    - Contact two airlines simultaneously for price quotes.
    - Buy a ticket if the quote is at most $300.
    - Buy the cheapest ticket if both are above $300.
    - Buy a ticket if the other airline does not give a timely quote.
    - Notify client if neither airline provides a timely quote.

Idea

- Start from a (functional) core including only concurrency
- Hierarchical structure: larger components by composition
- Few basic composition mechanisms (combinators)

Sites

- Site: basic service or component (the only concept!)
- Combinators for integrating sites
- Data types, processes, ... are programmed via sites

Sites

- A site is called like a procedure with parameters.
- Site returns at most one value.
- The value is published.

ORC programs

- Orc program has
  - a set of definitions
  - a goal expression
- The goal expression is executed.
- Its execution calls sites, publishes values.

ORC expressions

- Site call

  ```
  include "search.inc"
  Google("Pippo")
  ```

  call site Google with parameter Pippo, and publish the result.

- Composition

Composition

- Symmetric `f | g`

  do f and g in parallel

- Sequential `f >x> g`

  for all x from f do g

Composition

- Pruning `f <x< g`

  for some x from g do f

- Otherwise `f ; g`

  if f halts without publishing do g

Site calls: examples

- Prompt("What is your name?")
- 2 + 3
- true && false
- Println("Hello World")
  (publishes a signal - value with no info)

Other sites

- + −   *   && || =
- Println, Random, Prompt, Email ...
- Mutable Ref, Semaphore, Channel, ...
- Timer
- External Services: Google Search
- Any Java Class instance ...

Symmetric composition

```
f | g
```

- Evaluate f and g independently.
- Publish all values from both.
- No direct communication or interaction between f and g.

Example

- Call Bing and Google simultaneously
- Publish results from both (0, 1 or 2)

```
Bing(query) | Google(query)
```

- Ex. with Prompt site:

```
Prompt("Choice 1:") | Prompt("Choice 2:")
```

Sequential Composition

```
f >x> g
```

- Execute f and g in parallel
- All values published by f are passed to g through x

Example

- Get the results from Yahoo and Google, Filter both

```
(Bing(query) | Google(query)) >x> Filter(x)
```

- Example with Prompt:

```
(Prompt("Choice 1:") | Prompt("Choice 2:"))
   >x> x
```

Pruning

```
f <x< g
```

- Execute f and g in parallel
- Site calls which need x are suspended
- When g returns a first value
  - Bind it to x
  - Kill g
  - Resume suspended calls

Example

- Get the results from Bing or Google (only first answer is taken) and Filter

```
Filter(x) <x< (Bing(query) | Google(query))
```

- Example with Prompt:

```
x <x<
   (Prompt(" Choice 1:") | Prompt("Choice 2:"))
```

Otherwise

```
f ; g
```

- Execute f
- If f halts without publishing, then do g
- Expression halts if
    - its execution can take no more steps
    - all called sites have respondend or will never respond

Fork-Join

- Call Bing and Google in parallel
- Get their results as a tuple, when both responds

```
((b,g) <b< Bing(query))
       <g< Google(query)
```

Example

- Call Google only if Bing will never respond (site must be helpful)

```
Bing(query) ; Google(query)
```

Fundamental sites

- Ift(b), Iff (b):   boolean b
  signal if b is true/false; silent otherwise.
- Rwait(t): integer t, t $\geq$ 0
  signal t time units later.
- stop :
  never responds.
- signal:
  returns a signal immediately.

Example

```
Prompt("Value") >> stop ; Println("Stopped!")
```

- Output: “Stopped”

Guarded commands

```
Ift(b) >> c   |
Ift(b') >> c' |
Ift(b'') >> c''
```

```
3 >x>
(Ift(x :> 1) >> Println(">1") >> stop
 |
 Ift(x<=3) >> Println("<=1") >> stop
)
```

Function definition

```
def QueryLoop(query, t) =
   Google(query)
   >x> Println(x)
   >> Rwait(t)
   >> QueryLoop(query,t)
```

```
def metronome(t) =
   signal
 | Rwait(t) >> metronome(t)
```

With clauses and lists

```
def Sum([]) = 0
def Sum(h:t) = h + Sum(t)

Sum([1,2,3,3])
```

```
each(inviteList)
     >address>
     Email(address, invite)
```

Fibonacci

```
def Fib(0) = 1
def Fib(1) = 1
def Fib(n) = if (n <: 0) then 0
             else Fib(n-1) + Fib(n-2)
```

```
def H(0) = (1,1)
def H(n) = H(n-1) >(x,y)> (y,x+y)
def Fib(n) = if (n <: 0) then 0
             else H(n) >(x,_)> x
```

Java classes as sites

```
import class String = "java.lang.String"
val s = String("Pippo")
s.concat(" Baudo")
```

Channels

- factory site Channel():
  creates and publishes an asynchronous
  unbounded FIFO channel

Philosophers

```
def Fork(i, take, leave) =
    take.get() >>
    leave.get() >>
    Fork(i, take, leave)

def Phil(i, ltake, lleave, rtake, rleave) =
    Println(i + "think") >>
    ltake.put(1) >>
    rtake.put(1) >>
    Println(i + "eat") >>
    lleave.put(1) >>
    rleave.put(1) >>
    Phil(i, ltake, lleave, rtake, rleave)

def list(i) =
    if (i >= 0) then (i | list(i-1)) else stop

def Sys(i) =
   val take = Table0(i, Channel)
   val leave = Table0(i, Channel)
   list(i-1) >j> (Fork(j, take(j), leave(j)) |
                  Phil(j, take(j), leave(j), take(i(j+1)%i), leave((j+1)%i)))
```