Call/Answer

Lukas Renggli

renggli@iam.unibe.ch, University of Bern
renggli@netstyle.ch, netstyle.ch GmbH

Problem

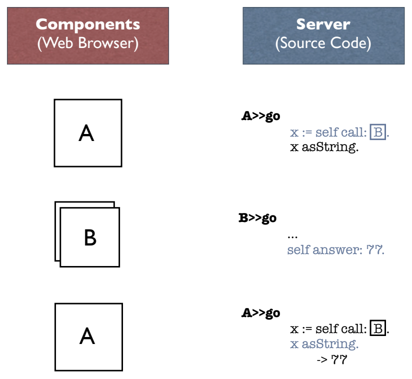
One Page

==

One script to run

Solution

- #call: aComponent
  - Transfer control to aComponent
  - aComponent will be given control
- #answer: anObject
  - anObject will be returned from #call:
  - Receiving component will be removed



```
A>>go
    x := self call: [B].
    x asString.
```

```
B>>go
    ...
    self answer: 77.
```

```
A>>go
    x := self call: [B].
    x asString.
        -> 77
```

Call/Answer

- Why do other frameworks not give you this simplicity?
- What is the magic behind call/answer?
- How is it implemented?

What is a continuation?

	Escaper
+	Context
=	Continuation

The Escaper

```
Object>>withEscaperDo: aBlock
   "Capture a starting-point and
   eventually return to it by evaluating
   the block that  is passed into aBlock."

   ^ aBlock value: [ :result | ^ result ].
```

break / continue

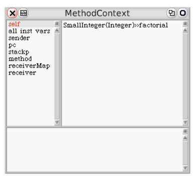
```
self withEscaperDo: [ :break |
   [ condition ] whileTrue: [
      self withEscaperDo: [ :continue |
         ...
         break value: 2.
         ... ] ] ].
```

The Context

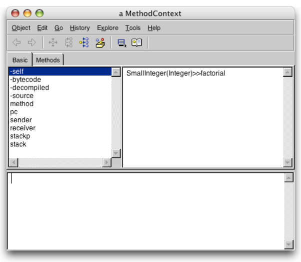
"A pseudo variable representing the current execution context. It contains references to the parent context, the receiver, the program pointer, the arguments and the current temporary variables."

```
thisContext
```

thisContext



```
SmallInteger(Integer)>>factorial
```



```
SmallInteger(Integer)>>factorial
```

The Continuation

Continuation class>>currentDo: aBlock

- Takes a one-argument block, that will be evaluated immediately with a continuation-object passed in.
- If this continuation is evaluated later on, it will abandon whatever calculation is in effect at that time and will instead resume the calculation that was in effect when the continuation was captured.

Simple Continuation

```
| result continuation |
result := Continuation currentDo: [ :cc |
   continuation := cc.
   false].
result
   ifFalse: [ continuation value: true].
self assert: x.
```

Examples

- Control structures
- Exception handling
- Non-local returns
- Co-routines
- Generators
- Web application flow

Seaside & Continuations

```
send-suspend: page-builder
  return continuation: [ :cc |
    cc-url := create unique url.
    register cc-url in server: [ :request |
      cc value: request ].
    html := build html with
      page-builder and cc-url.
    send response html.
    terminate process ].
```

#call:

```
WAComponent>>call: aComponent
  ^ Continuation currentDo: [ :cc |
    self
      show: aComponent
      onAnswer: cc ].
```

#show:onAnswer:

```
WAComponent>>show: aComponent
  onAnswer: aContinuation
  | delegation |
  delegation := WADelegation new
    delegate: aComponent.
  aComponent
    onAnswer: [ :value |
      delegation remove.
      aContinuation value: value ].
  self addDecoration: delegation.
```

Summary

Seaside provides a high abstraction over HTTP and programmers don't need to be aware of continuations that it is using to archive this.

Further Reading

- Stéphane Ducasse, Adrian Lienhard, Lukas Renggli, Seaside – Multiple Control Flow Web Application Framework
- Avi Bryant, HREF Considered Harmful http://www.cincomsmalltalk.com/userblogs/avi/blogView?searchCategory=continuations
- Lukas Renggli, Seaside Tutorial:
  - Questions: 36 – 40