An Introduction to Smalltalk for Objective-C Programmers

O'Reilly Mac OS X Conference
October 25—28, 2004

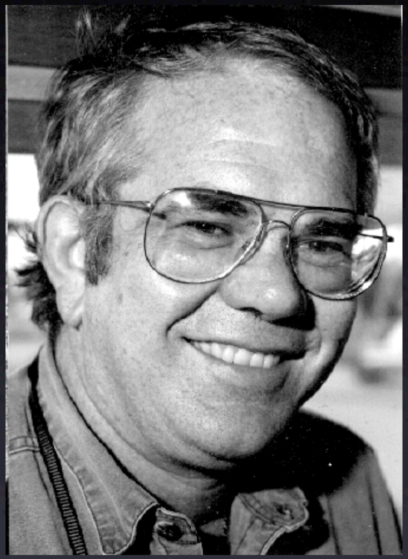
Philippe Mougin - pmougin@acm.org
http://www.fscript.org

STELAU
IT Management & Consulting

What you will get from this session

- An understanding of the basic principles of Smalltalk.
- The ability to read code.
- An overview of Smalltalk options available on Mac OS X.

Objective-C


Brad Cox

"Objective-C is a hybrid language that contains all of C language plus major parts of Smalltalk-80."

Objective-C: `[playList addSong:aSong]`

Smalltalk: `playList addSong:aSong`

Xerox Palo Alto Research Center

Alan Kay

Dan Ingalls

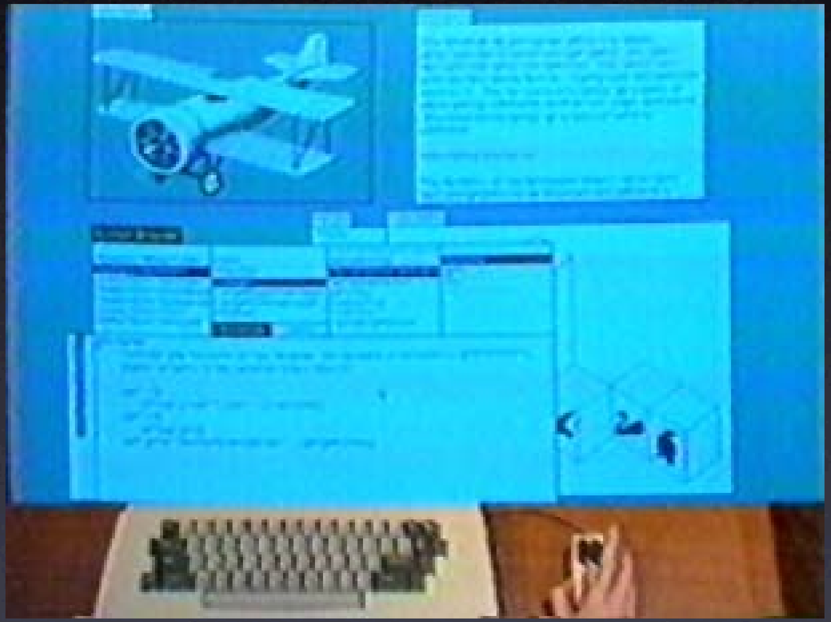
Adele Goldberg

Ted Kaehler

Steve Jobs meets Smalltalk

- "They showed me 3 things: graphical user interfaces, object oriented computing and networking."

Steve on a chair


Smalltalk on a Xerox Alto III

- "I only saw the first one which was so incredible to me that it saturated me..."
- "If I'd only stayed another 20 minutes!"
- "It took really until a few years ago for the industry to fully recreate it, in this case with NeXTSTEP."

Smalltalk

- ANSI standard approved in 1998: *ANSI INCITS 319-1998*
- A pure object language: everything is an object

> "Smalltalk's design is due to the insight that everything we can describe can be represented by the recursive composition of a single kind of behavioral building block that hides its combination of state and process inside itself and can be dealt with only through the exchange of messages." *Alan Kay*

- Message sending is (almost) the only control structure.
- Interactive environment.
- Simplicity and power.

Smalltalk-Based Environments

Ambrai Smalltalk
Dolphin Smalltalk
F-Script
GemStone/S
GNU Smalltalk
Little Smalltalk
Smalltalk/JVM
LSW Vision-Smalltalk
OOVM/Resilient Embedded Smalltalk
Object Studio Smalltalk
MicroSeeker PIC/Smalltalk
Pocket Smalltalk
Public Domain Smalltalk (PDST)

Sharp #Smalltalk
Slate
Smalltalk MT
S#
Smalltalk/X
Squeak
StepTalk
Strongtalk
Talks2
VisualAge Smalltalk
VisualWorks
Vmx Smalltalk
Zoku

ObjC/Smalltalk

Objective-C
message sending
control structures (`if`, while, for etc.)
explicit typing
objects
other data types
compile/link/run
reference counting

ObjC/Smalltalk

Smalltalk
message sending
objects
interactive system
garbage collector

Message sending syntax

Objective-C	Smalltalk
`[playList play]`	`playList play`

Unary message

Objective-C	Smalltalk
`[playList addSong:aSong]`	`playList addSong:aSong`

Keyword message

Message sending syntax

Objective-C	Smalltalk
`[object1 compare:object2] == NSOrderedDescending`	`object1 > object2`

Binary message

Message sending syntax

Objective-C	Smalltalk
`[[playList firstSong] artistName]`	`playList firstSong artistName`

Message sending syntax

Smalltalk	Objective-C Equivalent ?

`object1 method1:object2 method2`	`[[object1 method1:object2] method2]`
	or
	`[object1 method1:[object2 method2]]`

Message sending syntax

Smalltalk	Objective-C Equivalent ?
`(object1 method1:object2) method2` →	`[[object1 method1:object2] method2]`
	or
`object1 method1:object2 method2` →	`[object1 method1:[object2 method2]]`

Message Precedence:

1. Unary messages
2. Binary messages
3. Keyword messages

Base Library

Cocoa	Smalltalk
`class`	`class`
`copy`	`copy`
`isEqual:`	`=`
`hash`	`hash`
`isKindOfClass:`	`iskindOf:`
`isMemberOfClass:`	`isMemberOf:`
`performSelector:`	`perform:`
`performSelector:withObject:`	`perform:with:`
`respondsToSelector:`	`respondsTo:`
`forwardInvocation:`	`doesNotUnderstand:`
`description`	`printString`
`objectAtIndex:`	`at:`
`replaceObjectAtIndex:withObject:`	`at:put:`

Powerful Object Model

A few examples:

Asks a class for its subclasses:	`myClass subclasses`
Renames a class:	`myClass rename:'GreatClass'`
Asks a class for all its instances:	`myClass allInstances`
Adds an instance variable to a class:	`myClass addInstVarName:'lastName'`
Changes an object into another one:	`object1 become:object2`
Evaluates code given in a string	`Compiler evaluate:aString`

There are also methods for dynamically creating new classes, adding or removing methods, getting all the current references to an object etc.

The meta class level is accessible and can be modified to experiment new things, implement new features etc.

Assignment Syntax

Objective-C	Smalltalk
`a = 1`	`a := 1`

Instructions separator

Objective-C	Smalltalk
`instruction1 ; instruction2`	`instruction1 . instruction2`

Cascade

Objective-C	Smalltalk

`playList addSong:s1;`
`playList addSong:s2;`
`playList play;`
	`playList addSong:1; addSong:s2; play.`

Syntax Details

Objective-C	Smalltalk
`@"I am a string"`	`'I am a String'`
`'z'`	`$z`
`@selector(setTitle:)`	`#setTitle:`
`YES, NO`	`true, false`
`/* I am a comment */`	`"I am a comment"`
`3.14`	`3.14`
`self, super`	`self, super`
`nil`	`nil`

Methods (accessors)

Objective-C	Smalltalk
`- (NSString*) title`	`title`
`{`	`    ^ title`
`    return title;`	
`}`	
`- (void) setTitle:(NSString*) newTitle`	`title: newTitle`
`{`	`    title := newTitle`
`    [newTitle retain];`	
`    [title release];`	
`    title = newTitle;`	
`}`	

^ means “return”.
No explicit typing.
A different naming convention for the setter (`title:` instead of `setTitle:`).
No typing and the garbage collector makes things simpler.

Local variables

Objective-C	Smalltalk
`-(void)switchValueWith:(NSMutableString*) aString`	`switchValueWith: aString`
`{`	`    |temp|`
`    NSString *temp;`	`    temp = self copy.`
`    temp = [self copy];`	`    self setString:aString.`
`    [self setString:aString];`	`    aString setString:temp.`
`    [aString setString:temp];`	
`}`	

Local variables are declared between vertical bars.
You can have several local variables: `|local1 local2 local3|`
Local variables are initialized to nil.

Control structures

- Smalltalk does not have a specific syntax for expressing control structures like **if**, **for**, **while** etc.
- How can we do any serious programming without such constructs ???

Code blocks

Objective-C	Smalltalk

`{`
`  instruction1;`
`  instruction2;`
`}`
	`[`
	`  instruction1.`
	`  instruction2.`
	`]`

Executes instruction1 and instruction2.

Creates and returns a block object containing instruction1 and instruction2.

To execute the instructions in the block, you send it the "`value`" message.

Code blocks

Objective-C	Smalltalk
`{`	`[`
`  id local1, local2;`	`  |local1 local2|`
`  instruction1;`	`  instruction1.`
`  instruction2;`	`  instruction2.`
`}`	`]`

Executes instruction1 and instruction2.

Creates and returns a block object containing instruction1 and instruction2.

To execute the instructions in the block, you send it the "`value`" message.

Blocks are objects

- They can be sent messages
- They can be assigned to variables.
- They can be stored into collections.
- They can be passed as argument or returned from methods.
- They can be archived.
- Etc.

Conditional evaluation

- Boolean objects implements a method named `ifTrue:`
- This method takes a block as argument.
- The method `ifTrue:` triggers the evaluation of the block if and only if the receiver is true.

Objective-C	Smalltalk

`if (a > b)`
`{`
  instructions
`}`
	`a > b ifTrue:`
	`[`
	  instructions
	`]`

Conditional evaluation

- Booleans implement `ifTrue:ifFalse:`
- This method takes two blocks as argument.
- It triggers the evaluation of the first block if the receiver is true, and the evaluation of the second block if the receiver is false.

Objective-C	Smalltalk

`if (a > b)`
`{`
  instructions
`}`
`else`
`{`
  instructions
`}`
	`a > b ifTrue:`
	`[`
	  instructions
	`]`
	`ifFalse:`
	`[`
	  instructions
	`]`

Repetitive evaluation

- Blocks implement `whileTrue:`, a method that takes another block as argument.
- The block that receive the `whileTrue:` message repeatedly evaluates itself and, if the termination condition is not met yet, evaluates the argument.

Objective-C	Smalltalk

`while (a > b)`
`{`
   instructions
`}`
	`[a > b] whileTrue:`
	`[`
	   instructions
	`]`

Blocks can have arguments

- Each argument name is specified after a colon, at the beginning of the block. A vertical bar ends the argument list. Example: `[:a :b | a + b]`
- A block is evaluated by sending it an appropriate value... message.

```
['hello'] value        => 'hello'

[:a | a class] value:'aString'    => String

[:a :b | a + b] value:2 value:4    => 6
```

Repetitive evaluation

- Integers implement `to:do:`, a method that takes a number and a block as arguments.
- The block is evaluated for each integer between the receiver and the first argument (both included).

Objective-C	Smalltalk

`for (i = 1; i <= 100; i++)`
`{`
   instructions using i
`}`
	`1 to:100 do:`
	`[:i|`
	   instructions using i
	`]`

do:

- Collections implement `do:`, a method that takes a block as argument.
- The block is applied to each element of the collection.
- Example - Increase the volume of each song by 10%:

```
songs do:[:s| s raiseVolume:s volume * 0.1]
```

select:

- Collections implement `select:`, a method that takes a block as argument.
- This method returns a new collection containing only the elements in the receiver which cause the block to evaluate to true.
- Example - Creates a new collection containing the songs whose length is greater than 240 seconds:

```
songs select:[:s| s length > 240]
```

fork

- Blocks implement `fork`, a method with no arguments.
- This method evaluates the receiver in a new thread.
- Example - Play a song in a new thread:

```
[mySong play] fork
```

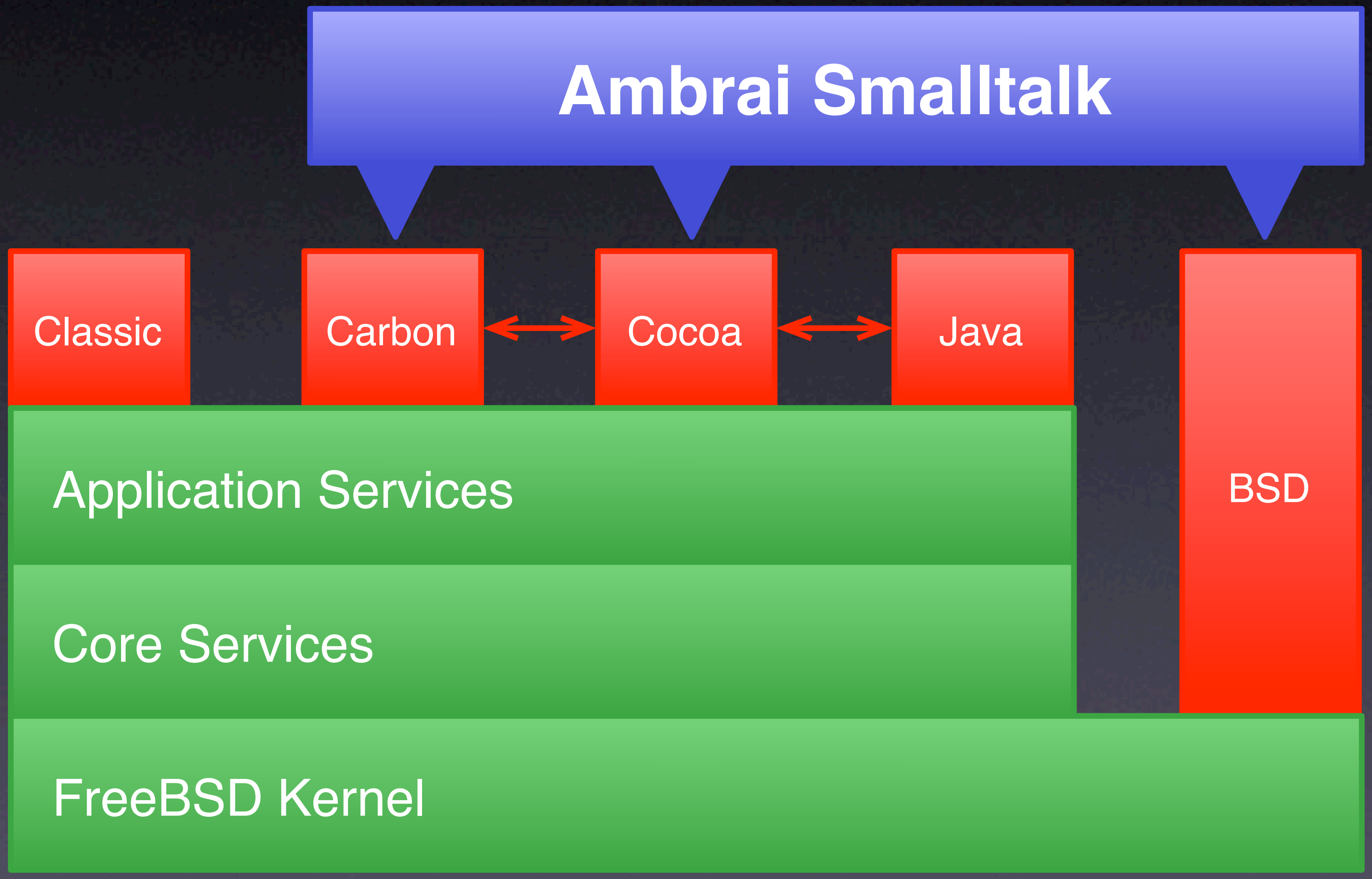
Other control structures

- Smalltalk provides many other powerful block-based methods in order to deal with collections, exceptions, multi-threading etc.
- Because no specific syntax is required, you can very easily implement your own “control structures” in terms of messages and blocks.

On Mac OS X

- Squeak
- VisualWorks
- Ambrai Smalltalk
- GNU Smalltalk
- JVM based Smalltalks
- F-Script

Ambrai Smalltalk



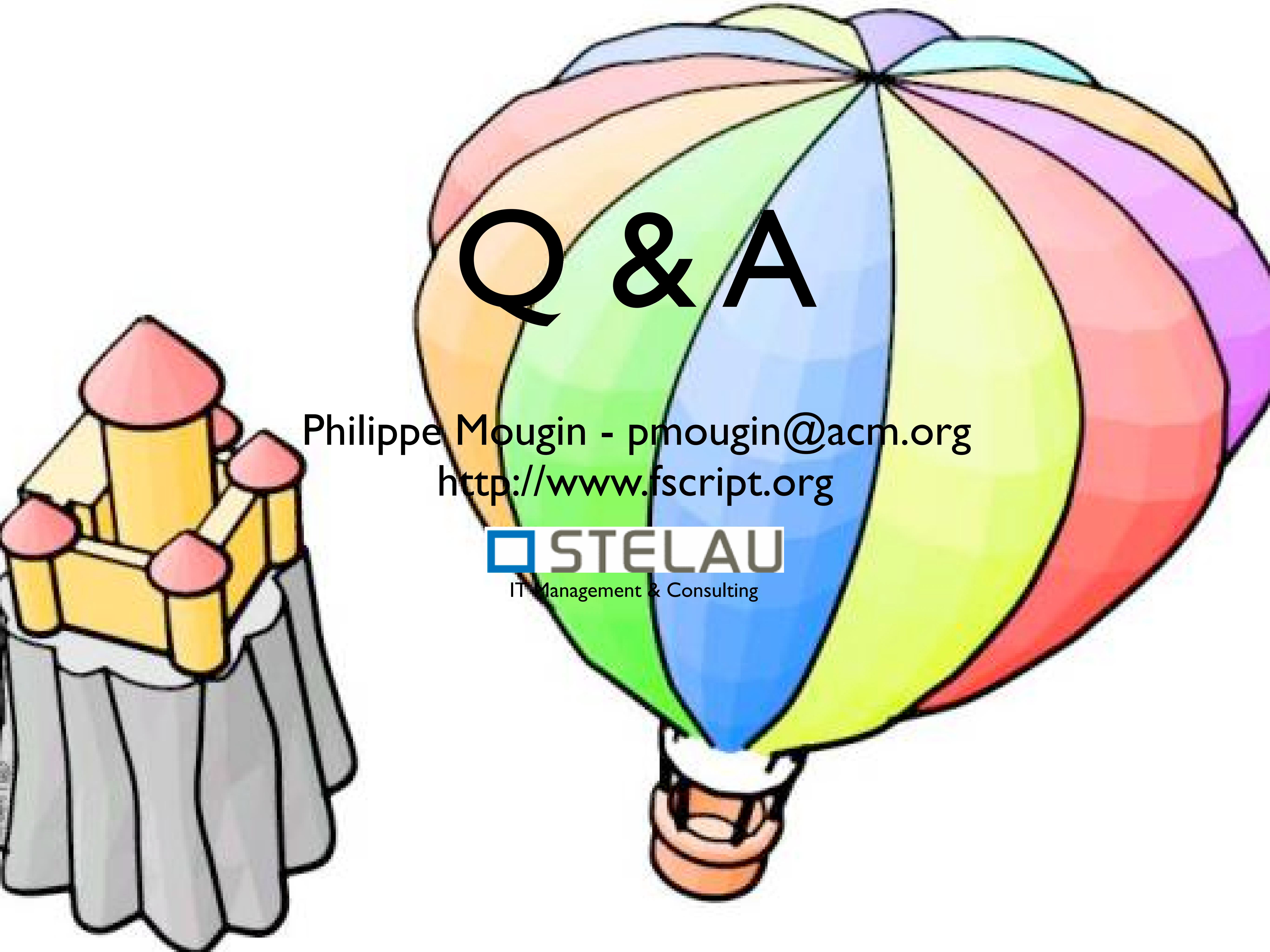
F-Script

- A lightweight open-source **interactive and scripting layer** using Smalltalk syntax and concepts.
- Built from the ground up for Cocoa.
- Let you interactively instantiate, explore and send messages to your Objective-C objects.
- Powerful graphical object browsers: explore and manipulate objects without writing code.

F-Script

- Integrated object query language capacities.
- The F-Script object model **is** the Cocoa object model. No bridging involved.
- Easy to embed into your own Cocoa application (framework + IB palette are provided).
- Can be dynamically injected into any existing Cocoa application.

http://www.fscript.org



Q & A

Philippe Mougin - pmougin@acm.org
http://www.fscript.org

STELAU
IT Management & Consulting