Cocoa Programming

David Chisnall

March 18, 2010

http://cs.swan.ac.uk/~csdavec/papers/SafariWebcast.pdf
The Story So Far

Understanding Cocoa
  What is Cocoa?
  Why Cocoa?

Cocoa Tricks
  Dictionaries for Flow Control
  Higher-Order Messaging
  Objects on the Stack
  isa-swizziling state machines

Portable Cocoa
  GNUstep Overview
  GNUstep Look and Feel
A Brief History of Objective-C

- 1980: Smalltalk-80 needed a really powerful workstation (512KB of RAM and a 5.8MHz CPU? Who can afford that?!)  
- 1983: Object-Oriented Pre-Compiler adds Smalltalk-like objects to C.  
- 1986: Objective-C adds Smalltalk-like syntax too.  
- 2002: Apple adds @try/@catch/@synchronized  
- 2007: Apple introduces Objective-C 2, the version that comes after Objective-C 4.0. Apparently this makes sense.
From OpenStep to Cocoa

- 1985: Steve Jobs leaves Apple to create the perfect workstation.
- 1988: NeXT Computer ships with first commercial Rapid Application Development environment.
- 1991: Sun joins NeXT to make a set of portable APIs.
- 1994: OpenStep specification published, GNUstep project starts implementing it.
- 1997: Apple buys NeXT, renames their OpenStep implementation Yellow Box.
- 2001: OS X 10.0 released, Yellow Box renamed Cocoa. GNUstep begins tracking Cocoa.
What is Cocoa?

Applications

Core Data Address Book ...

AppKit

CoreAnimation

CoreGraphics

Foundation WindowServer

libobjc

Core Foundation

libc

XNU

David Chisnall

Cocoa Programming
What is Cocoa?

Applications

Core Data | Address Book | ...

AppKit

Core Foundation

libobjc

Core Animation

CoreGraphics

WindowServer

Core Foundation

libc

XNU
What Makes Cocoa Special?

- Loose coupling everywhere (dynamic dispatch, notifications, KVC/KVO, and so on)
- Generic solutions (NSController, and so on)
- No magic (but lots of ‘sufficiently advanced technology’)

Dictionaries as Factories

```
- (void)registerHandlerClass: (Class)aClass
    forKey: (NSString*)aTag
{
    [handlers setObject: aClass forKey: aTag];
}

- (void)openTag: (NSString*)tag
    attributes: (NSDictionary*)attrs
{
    [self pushDelegate:
        [[[handlers objectForKey: tag] new] autorelease]];
    [delegate openTag: tag attributes: attrs];
}
```
Dictionaries as Switch Statements

```c
- (void)registerHandler: (SEL)aSel
  forException: (NSString*)except
{
    NSValue *boxedSel =
      [NSValue valueWithBytes: &aSel
       objCType: @encode(SEL)];
    [handlers setObject: boxedSel forKey: except];
    /* Or:
    [handlers setObject: NSStringFromSelector(sel)
                   forKey: except];
    */
}
```
The Story So Far
Understanding Cocoa
Cocoa Tricks
Portable Cocoa

Dictionaries as Switch Statements

```objective-c
@catch (NSException *e)
{
    SEL sel = @selector(ignore:);
    NSValue *boxedSel =
        [handlers objectForKey: [e name]];
    [boxedSel getValue: &sel];
    [self performSelector: sel withObject: e];
    // Or:
    // objc_msgSend(self, sel, e);
}
```
Why Dictionaries?

- Objective-C message sending depends on two things: the class and selector.
- Classes are objects.
- Selectors can be treated as objects.
What is Higher-Order Messaging?

- Based on the idea of higher-order functions.
- Messages that (appear to) take messages as arguments.
- Implemented with proxies.
- Some overhead, but make code much smaller.
HOM Examples

```objective-c
NSArray *uppercase = 
    [[array map] uppercaseString];
[[delegate ifResponds] handleArray: uppercase];
[[mediaPlayer inNewThread] playFile: aFile];
```

All of these work with EtoileFoundation (on OS X and GNUstep).
Implementing -ifResponds

```objc
@implementation IfRespondsProxy

// methodSignatureForSelector omitted

- (void)forwardInvocation: (NSInvocation *)anInvocation
{
    if ([object respondsToSelector: [anInvocation selector]])
    {
        [anInvocation invokeWithTarget: object];
    }
}
@end

@implementation NSObject (IfResponds)

- (id)ifResponds
{
    return [IfRespondsProxy proxyForObject: self];
}
@end
```
Things to Remember

- `-forwardingTargetForSelector:` can make proxies much faster.
- `-methodSignatureForSelector` must return something. This can be complicated.
- HOM code is relatively slow, but very concise.
Creating Objects on the Stack

```c
// Cache these values
Class class = [NSMutableIndexSet class];
size_t size = class_getInstanceSize(class);
struct { Class isa;
    char obj[size - sizeof(Class)]; } obj;
memset(&obj, 0, size);
obj.isa = class;
NSMutableIndexSet *set = (NSMutableIndexSet *)&obj;
[set init];
[set addIndex: 1];
// This will crash!
// [set dealloc];
```
Important Notes

- Don’t do this. Ever. Really.
- The performance gain is tiny for most objects.
- The code is really fragile.
- (Experts only) Design classes explicitly for stack allocation if you must do it.
State Machines

- Handle input.
- Process it depending on the current state.
- Useful for network protocols, and so on.
Implementing State Machines

```
- (void)handleInput: (char)a
{
    switch (state)
    {
        ...
        case STATE_A:
            // Do something with the input
            state = STATE_B;
            return;
        ...
    }
}
```

- State variable in class tracks state.
- Switch statements everywhere.
- Simple model, unmaintainable code.
Objective-C objects have a pointer to their class. The first instance variable, isa, holds this pointer. You can change this at run time.
State Machines with isa-swizzling

```c
- (void)handleInput: (char)a
{
    // Handle the input
    isa = [NewState class];
}
```

- One subclass per state.
- One method per action.
- Simpler code.
Caveats

- Using isa-swizzling breaks KVO (and potentially other things).
- Class layouts must be the same!
Portable Cocoa

▶ GNUstep tracks Cocoa changes.
▶ Request driven - classes and methods that people need get implemented first.
▶ Foundation will implement all of 10.4 in next release.
▶ AppKit will implement all of 10.2 in next release.
▶ Some bits of 10.6 are already working in both.
What is GNUstep?

Applications

Core Data  Address Book  ...

AppKit

Foundation  Cairo

libobjc  X.org / GDI

libc

Linux / FreeBSD / WinNT / ...

David Chisnall  Cocoa Programming
Is that all?

Applications

- Core Data
- Address Book
- ... (ellipsis)

AppKit

- Foundation
- Cairo

libobjc

- X.org / GDI

libc

Linux / FreeBSD / WinNT / ...

David Chisnall

Cocoa Programming
Bean requires OS X 10.4 or newer.
## Objective-C 2 Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>GCC</th>
<th>clang ObjectiveC2</th>
<th>clang libobjc2</th>
<th>libobjc2 nonfragile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-cost EH</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OS X 10.5 runtime APIs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Properties</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fast Enumeration</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Protocol introspection</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fast Proxies</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Non-fragile ivars</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Object planes</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Garbage Collection</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
</tbody>
</table>

Clang + libobjc2 has feature parity with iPhone runtime