Objective-C is Fun *

Adam Fedor

Objective-C is a language based upon C, with a few additions that make it a complete, object-oriented language. Why do I think Objective-C is fun? Precisely because of this emphasis on simplicity. Absolutely nothing was added to the language unless it really made a big improvement on the useability of Objective-C.

Fun, of course, can also lead to danger. And there is plenty to beware of with Objective-C. Since Objective-C doesn't try to improve the C language, you have to deal with all the faults and caveats of C. In addition, the philosophy of Objective-C is to allow for a great deal of flexibility, leaving the programmer to watch out for potential problems rather than forcing various restrictions.

For instance, Objective-C provides an object definition that is completely untyped. This is the id variable type, which stands for any object. With this type, you can send a message to any object, without knowing at all what the object is or what it can do.

Like C, this flexibility allows you to perform great tricks, while forcing the programmer to introduce his own conventions to prevent errors. But, ahh, there is fun in danger, isn't there?

Note: In the following article, I describe a particular variant of Objective-C known as OpenStep (or GNUstep, the free implementation of OpenStep). This is by far the most popular implementation of Objective-C (there is no standard Objective-C language yet ...).

*This article originally appeared in cscene, and Adam Fedor has permitted us to publish here under GNU FDL. — **FSM**

1 An Introduction

In truth, there is actually only one syntax addition to the language — the syntax for sending a message to an object:

```
[anObject doSomethingWith: anotherObject];
```

This statement simply sends the message doSomethingWith: to anObject, with an argument of anotherObject. An Objective-C message call can be used anywhere a C statement can be used, such as in a conditional statement or within another Objective-C message. On compilers that support it, Objective-C messages can be mixed in C++ code, allowing the programmer to pick and choose the best aspects of either language.

In addition, there are some keywords that have been added to allow for the definition and implementation of classes, such as the definition for the class of which anObject is a member:

```
@interface MyClass : NSObject
{
  int aVariable;
  id  subObject;
}

+ alloc;
+ defaultObject;

- init;
- (int) doSomethingWith: (id)anotherObject;

@end
```

Here, @interface tells the compiler we are defining an interface for the class named MyClass, which is a subclass of the class NSObject. NSObject is the root class used by GNUstep (The traditional Objective-C root class Object, is not used at all in GNUstep, although it is available). The root class is not a subclass of any other class.

Although it is not required that a class descend from a root class like NSObject, this hierarchy is so firmly ingrained in the language, that it is rare to see classes that are a root class by themselves. The root class contains some of the most basic and often used functionality, which gives you the advantage of knowing that all your classes will responds to these basic messages. Out of the 300 or so classes in the GNUstep core library, only two are root classes.

The curly brackets after the class definition enclose the data that is encapsulated in the class. These variables are called instance variables, because they are variables that belong to an instance of a class.

The rest of the interface defines the methods implemented by the class. A method is a description of a message a class responds to. A '+' indicates a class method. Class methods are often called factory methods, because typically a factory method is used to create or manufacture instances of the class. The most popular class method is alloc, which allocates space for an instance of the class (but does not initialize the class — that is usually reserved for the instance method init). A '-' indicates an instance method. For example, the doSomethingWith: method defined above takes one argument, the object anotherObject, and returns an integer. Class messages can only be sent to a class. Instance messages can only be sent to an instance of a class.

2 Dynamic Binding

Objective-C is a language that implements true dynamic binding (which is required for a language to be truly object-oriented). This means that messages sent to an object aren't bound to a specific function implementation in a specific class until the program is actually run. Stating this another way, the programmer does not know how an object will react to a specific message until the program is actually run.

Dynamic binding comes about because in every Objective-C program, there is a runtime that works behind the scenes to connect a specific message with a specific object. Every time a message like doSomethingWith: is sent to an object, the Objective-C runtime looks up the definition of the class of the object, finds the function (method) that corresponds to the message doSomethingWith:, and then calls that function.

It may seem like this type of lookup could really slow a program down. In practice, however, most runtimes are optimized for this sort of lookup, and in general, it can be shown that message lookup is often not the dominant time factor of a program (particularly in GUI programs). Anywhere that time is critical, it's possible in Objective-C to pre-bind a message to its implementation, thus avoiding the expensive message lookup.

An interesting aspect of a method definition, is that once a method is defined, that method name and definition are, in a sense global, and can be used anywhere. For instance, say I had an object newObject that did not recognize the doSomethingWith: message. Well I could send the message to the object anyway, and the compiler would not complain about it:

```
  id newObject;
// could be any object;
  newObject = [[NewObject alloc] init];
// create and initialize the object
  [newObject doSomethingWith: anotherObject];
// send it a message.
```

Only when the program was run would the Objective-C runtime discover that newObject did not recognize

the doSomethingWith: message. However, instead of immediately issuing an error, the runtime instead sends another message to newObject, called the forwardInvocation: message. This gives newObject a chance to handle messages sent to it that it doesn't understand. In this case, if newObject knew about anObject, it could forward the message on to anObject and let it handle the message:

```
- (void) forwardInvocation: (NSInvocation*)\
anInvocation
{
  if ([anObject respondsToSelector: [anInvo\
cation selector]])
    return [anInvocation invokeWithTarget: \
anObject];
  else
    return [self doesNotRecognizeSelector: \
[anInvocation selector]];
}
```

Here, the variable self refers to the object which received the message (similar to the this variable in C++). anInvocation is another object which contains all the information about the message that was sent, including the name of the message (the selector) and any arguments.

This type of usage is called delegation, and it is a powerful method of implementing various different constructs such as multiple inheritance, journaling, and dispatching messages to dynamically loaded code.

Incidentally, there are proper ways to handle unrecognized message, using Protocols, that allow for compile time checking and more robust code. Most programmers avoid defining un-typed objects unless it's really useful. A better way to write the first code segment would be:

```
  NewObject *newObject;
// newObject will be an instance of
// the NewObject class
  newObject = [[NewObject alloc] init];
// create and initialize the object
  [newObject doSomethingWith: anotherObject];
```

In this case, the compiler would issue a warning stating that newObject does not respond to the message doSomethingWith:.

```
\section{Implementation}
```

I've already given an example of how to implement a method with the forwardInvocation: method. That example wasn't entirely correct. In Objective-C, a method implementation must be associated with a certain class by enclosing the implementation between the keywords @implementation and @end.

```
@implementation MyClass

- (int) doSomethingWith: anotherObject
{
 return [anotherObject multiply: 3 by: 4];
}

@end
```

The @implementation keyword, is a counterpart to the @interface keyword (Note that it is not necessary to indicate that MyClass is a subclass of NSObject. The compiler already knows this from the interface definition). You can implement as many methods as you like within an @implementation block (even methods that were not described in the class interface). Methods that are implemented in the implementation section but are not described in the interface become, in essence, private methods that can only be used by that class and not by another class.

3 Power through Customization

One of the most frustrating aspects of using classes written by other people is that these classes often don't contain some key functionality that you

need. One way to add this functionality (assuming you don't have the source to the original class, or don't want to change it for some other reason), is to simply make a subclass and implement the methods you need in that class. But this creates another problem. Now you have to tell everyone else to use your particular subclass and not the original class. Or what if someone else has already subclassed this class and added different functionality. How do you combine the two subclasses?

Objective-C provides a very simple and elegant way to add functionality to an existing class using Categories. A category is defined simply with an @interface line and the name of the category (e.g. MyAdditionalMethods):

```
@interface MyClass (MyAdditionalMethods)
- (int) doSomethingWith: thisObject andThen\
With: thatObject;
@end
```

The additional methods are then implemented in a complementary @implementation section. There is one restriction, however. You can't define additional instance variables, which would change the amount of data held by the class. Other than this restriction, the things you can do with categories is incredible (and bizarre). For instance, if a method is defined both in the main class interface and a category, the category implementation wins. A message sent to the class will use the category implementation and ignore the original class implementation. If a method is defined in two separate categories, the implementation that is used is indeterminate, since you never know which category will be loaded first. In a situation where dynamic loading is used, this sort of behavior can create all sorts of headaches for a security conscious programmer ...

Most often, however, categories are used to implement additional functionality that may be useful to only one part of the program. For instance, some additional methods may be defined to allow a class to better interact with the GUI section of the program, or the distributed objects section of the program.

4 And More...

I've only touched on a few of the features of Objective-C. I also haven't answered a lot of typical first-time Objective-C user questions, like why is the allocation of space for a class (alloc) separated from the initialization of the class (init)? How do you get rid of an object once your done with it (garbage collecting)? Future articles might tackle these questions.

To learn more, I'd encourage you to visit the GNUstep home page:

`http://www.gnustep.org`,

or the Apple Developers page at:

`http://developer.apple.com/techpubs/macosx servermacosxserver.html`

to look for more documentation on the Objective-C language and the OpenStep libraries.

[FreeBSD 4.5 is to Release] Daemon News will be producing a 4 CD set of FreeBSD 4.5 in Feb 2002. These will be made using the official FreeBSD project ISO images, packaged in a standard jewel case. It is reported Daemon News will be the official distributor of FreeBSD since 4.5 release.

We tested FreeBSD-4.4 in Dec 2001, both GNOME and KDE are included. It truly works "rock-solid" stable without X Window system installed, so it is quite a good option to recommend for running as Internet servers, but for desktop users, it is buggy if the X-Window system installed on, the system sometimes gets frozen unexpected. Hopefully the 4.5 Release will be stable to work with X Window for the desktop and workstation users.
