Flujo de control en iOS

Flow control in iOS

Franklin Hernández Castro[1]
Jorge Monge Fallas[2]

Fecha de recepción: 20 de enero del 2012
Fecha de aprobación: 19 de marzo del 2012

Hernández, F; Monge, J. Flujo de control en iOS. *Tecnología en Marcha*. Vol. 25, N° 5. Pág 73-78.

Este artículo cuenta con el aval de la Vicerrectoría de Investigación y Extensión del Tecnológico de Costa Rica

1 Diseñador industrial. Escuela de Ingeniería en Diseño Industrial, Tecnológico de Costa Rica. Teléfono: 2550-2598. Correo electrónico: franhernandez@itcr.ac.cr

2 Profesor de matemática. Escuela de Matemática, Tecnológico de Costa Rica. Teléfono: 2550-2703. Correo electrónico: jomonge@itcr.ac.cr

Resumen

El objetivo de este artículo es explicar los flujos de control que se usan en la programación de las aplicaciones en iOS, con el fin de resumir los aspectos más relevantes que se deben tomar en cuenta para programar una tarea a ser realizada por un dispositivo móvil del tipo iPhone o iPad. Debido a que el ambiente iOS es estrictamente orientado a objetos (OOP), los flujos de control no son obvios; además, los estándares de la firma Apple® definen patrones de diseño en el sistema que son altamente recomendados en este tipo de diseño. En este artículo se introducen algunos de ellos.

Palabras clave

iOS, *Model View Controller MVC*, Apps, iPad, iPhone, flujo de control, *AppDelegate*, *ViewController*, *Objetive-C*.

Abstract

This paper explain the control flows that are used by programming applications in iOS, trying to summarize the most important aspects to be considered by programming mobile devices like iPhone and iPad. Because iOS environment, is strictly a object-oriented one(OOP), control flows are not obvious, besides Apple® use design patterns highly recommended in this type of programming. Here we introduces some of them.

Key words

iOS, *Model View Controller MVC*, Apps, iPad, iPhone, control flow, *AppDelegate*, *ViewController*, *Objetive-C*.

Introducción

A pesar de que el lenguaje *Objetive-C* es un lenguaje basado en C, los flujos de control dentro de este ambiente son muy diferentes a los que esperaría un programador de C no habituado a este ambiente.

En la inmensa mayoría de los casos, los proyectos empiezan con un *template* ya definido por el ambiente de programación (*Xcode*). En la actualidad (iOS 5) se ofrecen siete de estas plantillas como posible inicio de una aplicación, con esto el ambiente de programación (*integrated development environment*, IDE) pretende generar los pasos comunes y necesarios para que una aplicación pueda trabajar correctamente. Sin embargo, los tipos de plantillas de inicio han variado de versión a versión de *Xcode*, por lo que no se recomienda usarlas en forma irreflexiva, sino más bien tratar de entender su funcionamiento.

Algunos aspectos propios de los patrones de diseño de Apple® son seguidos en forma estricta en estas plantilla y, por lo tanto, se apartan de las prácticas usuales en programación C en general.

El uso del *main* como iniciador del flujo de control, por ejemplo, solo se conserva como una curiosidad histórica, pues en la inmensa mayoría de las aplicaciones iOS, un pequeño *main* queda como el iniciador no modificado por el programador de todo el proceso.

Paradigma MVC (Model View Controller)

Para entender los flujos de control propuestos por Apple®, es necesario repasar por un momento el paradigma *Model View* ***Controller*** (MVC), que es la base de la propuesta en todas las plantillas de programación. Se trata de la separación, clara y definida de los tres componentes necesarios para llevar a cabo una tarea en los dispositivos:

1.View: es el módulo encargado de recibir los *inputs* del usuario (principalmente los gestos *touch*) y de desplegar los resultados que desencadenan estos gestos.

2.Model: es el módulo donde se hacen los cálculos necesarios o donde se obtiene la información para responder a estos gestos/*inputs*.

3.Controller: es el medio de comunicación entre los dos módulos anteriores.

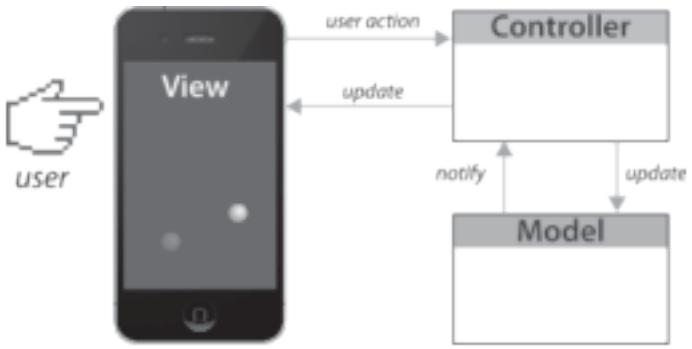

Figura 1. Definición de los componentes del patrón MVC.

En el ejemplo de la figura se desea tener unas bolitas flotantes en la pantalla; estas reaccionarían con la posición del iPhone, de modo que, cuando se inclina, las bolitas se desplazan "por gravedad" al lado que en ese momento está hacia abajo.

En este caso, el componente *view* se encarga de "pintar" las bolitas y reconocer la posición del iPhone (¿qué lado está hacia abajo?), a la vez que transmite esta posición al controlador, quien comunica esta información al modelo. El modelo, que son las bolitas (posiblemente una sola clase que se multiplica en el controlador), toma la información de la posición del iPhone, y recalcula su nueva posición, para luego comunicársela al controlador, que, a su vez, informa al *view*, el cual se encarga de redibujar las bolitas en su nueva posición.

En primera instancia, parecería (en este simple ejemplo) que es un paso de más poner el controlador entre el modelo y el *view*; sin embargo, la idea es mantener la interfaz completamente separada del modelo, de modo que el mantenimiento de la aplicación se facilita en gran manera.

Todo comienza en el appDelegate

En realidad, todo el proceso comienza en un archivo generado en la mayoría de las plantillas, denominado *appDelegate*. Este archivo, generalmente, obtiene el nombre de la aplicación más este apéndice. En el caso de que se nombre la aplicación del ejemplo "Gravedad", el nombre generado para el *appDelegate* sería Gravedad*AppDelegate*.

En iOS, el encabezado normal de una clase en lenguaje C se define en un archivo aparte con la extensión h (de *header*); el cuerpo de la clase, en cambio, con sus métodos y constructor, usa otro archivo con el mismo nombre anterior pero con la extensión .m. En una clase normal en C, ambos formarían parte del mismo archivo.

Al interior del *appDelegate* se ofrece (desde la plantilla) una serie de funciones previstas para responder a eventos. En este ejemplo, y en la mayoría de los casos, la más usada de estas funciones es la *didFinishLaunchingWithOptions*. Esta función se ejecuta justo en el momento en que la aplicación ha sido montada en el sistema operativo.

- (BOOL)application:(UIApplication *)application **didFinishLaunchingWithOptions**:(NSDictionary *)launchOptions

En esta función es donde se crean los objetos que darán cuerpo a la aplicación. Sin embargo, el objeto más importante que se origina al interior del *appDelegate* es el Controlador, el cual se genera de forma automática, a menos que se defina lo contrario.

Es decir, si se ha programado una clase tipo *ViewController*, es factible que se cree al interior de la función (*didFinishLaunchingWithOptions*) un objeto de esta clase y este se declare como el controlador inicial de la herramienta con las características deseadas.

El ViewController

Como se dijo, una vez que se corre el *appDelegate*, si no se ha hecho ya, se genera en forma automática el objeto *ViewController*.

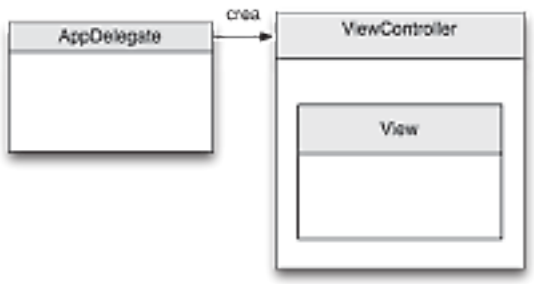

Figura 2. Relación entre el appDelegate, el controlador y el view.

Lo más importante de anotar en este caso es que todo controlador tiene, como una de sus propiedades, un objeto del tipo *view*. Es decir, que para mantener una adherencia al patrón de diseño MVC, la plantilla crea, en forma automática, un objeto de la clase *view* y lo declara como propiedad del controlador.

Como en el caso del mismo controlador, este objeto *view*, puede crearse automáticamente, en estos casos, el archivo no se hace visible en el *Xcode*; sin embargo, también se puede haber generado en forma explícita un objeto de la clase del tipo *view*, esto se haría en la función (*didFinishLaunchingWithOptions*) del *appDelegate*. Desde allí se puede haber creado un objeto de esta clase y declarado como la propiedad *view* asociada al controlador.

En la clase *ViewController* también se generan automáticamente una serie de funciones que responden a eventos. Algunas de ellas tienen que ver con la creación de la *view*:

- (void)viewDidLoad

- (void)viewDidUnload

- (void)viewWillAppear:(BOOL)animated

- (void)viewDidAppear: (BOOL)animated

- (void)viewWillDisappear:(BOOL)animated

- (void)viewDidDisappear: (BOOL)animated

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation

Este tipo de funciones, como su nombre lo indica, definen momentos específicos en los que el programador puede crear objetos que le sean útiles. Del mismo modo, puede recurrir a acciones cuando la *view* deja de estar presente (*viewDidUnload*, por ejemplo); estos casos son especialmente importantes en aplicaciones donde se navega entre *views*, es decir, si se tiene una aplicación en la que se pasa de un *view* (donde se definen preferencias, por ejemplo) a otro donde se realiza otra acción, se ejecutaría, en eventos como *viewDidUnload,* el pase de variables entre un escenario y el otro. Además, es en estas funciones que se crean los objetos que se encargarán de llevar a cabo los procesos deseados.

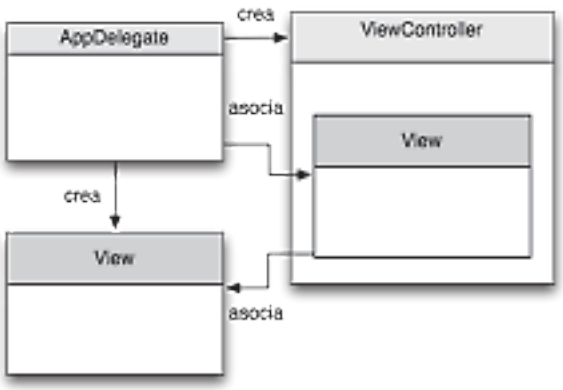

Figura 3. Relación entre el appDelegate, el controlador y el view cuando es creado por el appDelegate.

Aquí es donde se define, por ejemplo, el objeto u objetos que conforman el modelo.

Además, en el *ViewController* hay otras funciones para otros eventos:

- (void)didReceiveMemoryWarning

Donde se debe tomar las acciones del caso si se presentara esta situación. Un ejemplo típico es: ¿qué le sucede a la aplicación si en ese momento se recibe una llamada telefónica? En este caso, se utiliza este tipo de funciones tanto para "bajar" la aplicación como para reanudarla.

El ciclo

El ciclo del programa depende mucho del diseño de la aplicación. En primera instancia, en una aplicación típica, el ciclo va a depender de los *inputs* que se hayan definido. Es decir, si la aplicación tiene botones, estos llaman funciones definidas por el programador que actualizan valores en campos de texto, o redibujan objetos en el *View*. Esto sucede automáticamente al mover un *slider* o accionar un botón, por ejemplo.

El otro caso se da cuando una acción debe realizarse en forma continua. Por ejemplo, en animaciones o juegos se espera que las cosas sucedan en forma continua hasta que el usuario haga alguna acción que cambie las condiciones de funcionamiento.

En el ejemplo de las bolitas, se supone que las bolitas "caen" en forma automática hasta que el usuario mueve el iPhone, lo que ocasiona que las bolitas empiecen a caer en otra dirección.

En estos casos, es necesario llamar a una función tipo *NSTimer*, la cual, generalmente, se define en el Controlador (se programa dentro de la función *(void)viewDidLoad*). Esta función designa tanto un intervalo de tiempo como una función que será llamada en ese intervalo, ejemplo:

```
self.animationTimer=[NSTimer
scheduledTimerWithTimeInterval:

animationInterval
                        target:         self
                         selector:      @
selector(tic)
                         userInfo:      nil
                         repeats:       YES];
```

Es decir, en el Controlador se programa una función que debería ejecutarse cada cierto intervalo de tiempo, por ejemplo, treinta veces por segundo. Generalmente, esta función a ejecutarse se denomina "tic", lo cual recuerda el tic de los relojes.

En la función "tic", lo que se programa es la actualización de los datos. En el ejemplo de las bolitas, se diría que lo que hace la función tic es leer la posición del iPhone que, en última instancia, significa leer la aceleración en X y en Y de los osciloscopios. Luego manda estos datos a una función de actualización en los objetos del tipo Bola (es decir, en el Modelo). De esta forma estos objetos han actualizado la posición.

Concluida esta actualización lo que queda es decirle a la *View* que redibuje los objetos, lo cual se puede hacer en tres sitios distintos:

1. En los objetos mismos, en cuyo caso hay que mandar un "contexto gráfico" como parámetro, de modo que los objetos sepan donde se deben dibujar.
2. En esta misma función, en caso de que los objetos sean sub-*views,* solo se necesita actualizar su posición;.
3. En el objeto tipo *View*, en caso de que el usuario haya definido un objeto tipo *View* en forma propia, se marca al objeto *View* para que se redibuje con los datos actualizados (generalmente con la función [*self.view setNeedsDisplay*]), todo esto desde el controlador.

En este último caso, toda clase tipo *View* tiene una función llamada *(void)drawRect:(CGRect)rect*, y es desde aquí que se dibujan las cosas que se desean en este escenario.

Conclusión

El flujo de control en iOS no es comparable con un flujo de control en C estándar. Hay que tomar en cuenta los patrones de diseño de Apple® si se desea hacer un uso eficiente, tanto del ambiente de desarrollo *Xcode* como de los recursos de los aparatos.

El flujo de control comienza siempre en el *appDelegat*e y de ahí pasa al *ViewController*, desde donde se crea el modelo y se controla el módulo *View* que es, en última instancia, quien dibuja lo que sea necesario y recibe los *inputs* del usuario.

Bibliografía

Goldstein, N. & Bove, T. (2010). *iPad Application Development for Dummies*. Obtenido desde http://my.safaribooksonline.com/book/hardware-and-gadgets/9780470580271.

Lewis, R. (2010). *iPhone and iPad apps for Absolute Beginners*. Obtenido desde http://www.apress.com

Ray, J. (2011). *iPad Application Development*. Obtenido desde http://bookmoving.com/book/sams-teach-ipad-application-development-hours-sams-teach-hours-_51204.html