Docencia de la Programación Orientada a Eventos

Carlos Rioja del Río
Dept. de Lenguajes y Sistemas Informáticos
Universidad de Cádiz
11003 Cádiz
e-mail: carlos.rioja@uca.es

Resumen

Se presenta una vía de enseñar programación orientada a eventos o dirigida por eventos basándose en el conocimiento de la programación orientada a objetos (POO), mostrando la orientación a eventos como una extensión de la orientación a objetos.

1. Introducción

La enseñanza de una nueva metodología de programación supone para el alumnado el esfuerzo de cambiar la manera de pensar a la hora de abordar un problema. Por todos es bien conocido la dificultad que supone modificar o ampliar los esquemas de razonamiento de una persona si tiene otros esquemas distintos bien arraigados. Es el caso de los alumnos de los primeros cursos que una vez han conseguido trabajar de manera eficiente con la metodología de programación estructurada, se enfrentan al aprendizaje de la programación orientada a objetos, lo cual les hace cambiar por completo su manera de ver el universo que les rodea. Este problema se acentúa si la nueva metodología a aprender es la de la programación lógica o la programación funcional.

En definitiva, cualquier cambio que ha de producirse no en el conocimiento, sino en las vías de adquisición de conocimiento, confieren un gran esfuerzo por parte del alumno y del profesor, y normalmente algunas de las metodologías impartidas nunca llegan a asimilarse por completo por parte del alumno debido a que el alumno tiende a abordar los problemas siempre desde los mismo puntos de vista, aunque la solución obtenida exceda en dificultad a otras soluciones alternativas. Por tanto, es conveniente minimizar las dificultades que pueda encontrar el alumno a la hora de conocer una nueva filosofía de programación. En algunos casos, es una tarea compleja, como en los comentados anteriormente, poco tiene que ver la programación estructurada con la programación lógica o la orientación a objetos. Sin embargo, a la hora de enseñar programación visual o programación orientada a eventos, no debemos presentarla al alumnado como una novedad absoluta, más bien, como una evolución de la orientación a objetos, buscando el aprendizaje por analogía. Se presentan a continuación las vías para conseguirlo.

2. Contexto y Motivación

En la Ingeniería Técnica en Informática de Gestión de la Universidad de Cádiz, la asignatura principal en la que se enseña el paradigma de la programación orientada a objetos es obligatoria y se imparte en segundo curso. Así mismo, la asignatura de desarrollo rápido de aplicaciones y por tanto, la programación orientada a eventos es optativa para los cursos segundo y tercero.

Nos encontramos entonces con la posibilidad de que el alumno que se enfrenta a la programación orientada a eventos ya posea conocimientos del paradigma orientado a objetos, o que se encuentre cursando las dos asignaturas a la vez. En el primer caso el profesor puede y debe apoyarse en los conocimientos de POO adquiridos para presentar las novedades de la orientación a eventos. Y en el segundo caso es necesario un seguimiento del avance en ambas asignaturas para reforzar, si cabe, el aprendizaje de las partes comunes de ambas metodologías, que son muchas e importantes.

Por supuesto, se podría enseñar programación orientada a eventos obviando cualquier relación

posible con la orientación a objetos, pero esto aumenta el esfuerzo a realizar por el alumno, que está recibiendo dos metodologías como completamente distintas, cuando una es pilar en el desarrollo de la otra.

3. El paradigma de la orientación a objetos.

En un sistema orientado a objetos, el software se organiza como un conjunto finito de objetos que contienen tanto datos como operaciones y que se comunican entre sí mediante mensajes [1].

Un alumno con conocimiento de la programación orientada a objetos debe ser capaz de distinguir las clases y objetos de un determinado problema y describir la importancia de la herencia en los programas orientados a objetos. Mediante la *herencia*, una clase puede heredar atributos y métodos de otra, a la que llamaremos superclase, de esta manera podemos formar jerarquías de clases.

La *encapsulación* y la *ocultación* de datos son características fundamentales que nos ayudan en el diseño de programas orientados a objetos. De esta manera, las clases encapsulan un conjunto de datos y operaciones comunes en un mismo nivel de abstracción. Gracias a la ocultación de datos podemos separar la información necesaria para la implementación del programa de la interfaz con la que trabajará el usuario.

Decimos que una clase que presenta interfaz para varias clases es de tipo *polimórfico* [2]; el polimorfismo es también uno de los elementos fundamentales de la POO.

4. La programación orientada a eventos

En las herramientas de desarrollo rápido de aplicaciones, la escritura de un programa se basa en el uso de componentes prediseñados que se colocan en formularios o ventanas y se ajustan a las necesidades del problema mediante el establecimiento de las propiedades de los componentes en cuestión.

En la programación orientada a eventos, la línea de ejecución del programa no está dictaminada de antemano, no conocemos cuáles serán las líneas de código que se ejecutarán en cada caso. Es el usuario o el sistema los que determinan la ejecución de las funciones como respuesta a eventos provocados por ellos.

Un evento es una acción que es reconocida por un objeto y que normalmente es provocada por el usuario al interactuar con la interfaz del programa (la pulsación de un botón del ratón, la pulsación de una tecla, etc.). Muchos objetos tienen predefinidos un conjunto de eventos que pueden reconocer, si uno de ellos ocurre, se ejecuta un manejador de evento (función) como respuesta, por lo tanto, una aplicación para el sistema operativo en realidad lo que hace es ejecutar funciones para tratar los distintos eventos que se vayan produciendo. De ahí viene la analogía con la POO, en un sistema dirigido por eventos, los componentes prediseñados que usamos pueden verse como objetos, pues tienen una serie de propiedades o atributos y un conjunto claramente definido de funciones miembro o métodos.

El flujo interno de una aplicación dirigida por eventos puede describirse a través de los siguientes pasos:

1. La aplicación carga en memoria y visualiza el formulario de inicio. (Este paso puede variar entre distintas herramientas RAD).
2. La aplicación espera hasta que se produzca un evento.
3. Se determina el tipo de evento producido.
4. Se determina el objeto afectado por el evento.
5. Se ejecuta el manejador de evento correspondiente.
6. La aplicación vuelve al paso 2.

Un alumno con conocimiento de la programación orientada a eventos debe ser capaz de distinguir, para cada problema particular, la interfaz gráfica del usuario, los componentes que la compondrán, las propiedades de los mismos, y los manejadores para los posibles eventos que puedan producirse en el sistema. El desarrollo de un programa orientado a eventos se simplifica en estos tres pasos. Sin embargo, el aprendizaje del entorno de desarrollo puede ser deficiente si no mostramos la orientación a eventos como una derivación de la orientación a objetos.

5. Establecimiento del paralelismo

Una vez el alumno conoce los conceptos fundamentales de la orientación a objetos tales como clase, objeto, polimorfismo, herencia, etc, es el momento de presentarle cómo en una herramienta de desarrollo rápido tenemos una serie de clases de las cuales se derivan subclases con los componentes que vaya a utilizar en cada aplicación.

Utilizaremos para ello como herramienta de desarrollo rápido Borland C++ Builder, cuyo lenguaje de programación embebido es C++. Una aplicación desarrollada con Borland C++ Builder, al igual que en el resto de lenguajes visuales, está formada por una serie de formularios que contienen componentes y responden a los eventos producidos en esos formularios. De esta manera, podemos considerar el formulario como el pilar fundamental sobre el que se asienta cualquier aplicación orientada a eventos. En el caso de Borland C++ builder, cada uno de los formularios se corresponde con tres archivos tales como Unit.cpp, Unit.h, Unit.dfm. El archivo .cpp contiene el código C++ con los manejadores de eventos necesarios para la correcta ejecución del programa. El archivo .h contiene inclusiones de otros ficheros de cabecera, declaraciones, prototipos de funciones, constantes, etc. Y el archivo .dfm tiene la descripción gráfica del formulario.

En este punto debemos destacar que el compilador coloca en el archivo .h mucha información de manera automática, y esta información suele pasar desapercibida para el alumno. Es en este caso en el que debemos mostrar la importancia de conocer realmente qué ocurre cuando diseñamos una aplicación con formularios y cómo éstos son objetos de una clase derivada de la clase TForml la cual nos proporciona el entorno de desarrollo.

Cuando estamos diseñando una aplicación, podemos observar en el archivo .h cómo ese formulario con el que estamos trabajando se implementa como un puntero a una clase derivada de TForm, esto ocurre de la siguiente manera:

```
class TForm1 : public TForm
{
__published:
private:   // User declarations
public:    // User declarations
    __fastcall
TForm1(TComponent* Owner);
};
//---------------------------
extern TForm1 *Form1;
```

Podemos observar en el código cómo automáticamente en el archivo .h se deriva la nueva clase TForm1, siendo Form1 el nombre del formulario que estamos diseñando. También se incluye una declaración de tipo externa para que el formulario pueda ser modificado por todos los archivos que incluyan este fichero de cabecera. La definición del objeto Form1 aparece en el fichero .cpp

Es importante evitar que el alumnado pase por alto estas inclusiones automáticas y se limite a seleccionar componentes y ubicarlos en los formularios. La realidad es bien distinta, cuando generamos nuevos formularios para nuestra aplicación, estamos derivando subclases de la clase madre TForm, proporcionada por Borland C++ Builder, y cuando se seleccionan componentes para incluirlos en un formulario, estamos añadiendo atributos a la clase derivada TForm1. A su vez, estos componentes son punteros a clases proporcionadas por el entorno de desarrollo y como tales, poseen propiedades y métodos.

6. En la práctica. Paso a paso.

Vamos a mostrar a continuación cómo a través de un ejemplo práctico podemos ir paso a paso mostrando aquellos aspectos del desarrollo rápido de aplicaciones que nos permiten establecer un símil con la programación orientada a objetos. Desarrollaremos una aplicación con un formulario que contiene dos componentes botones y un componente etiqueta que mostrará un cierto texto al usuario.

El primer paso es mostrar al alumno cómo en cuanto comenzamos a desarrollar la aplicación, el formulario principal que estamos usando es un puntero a una clase derivada tal y como hemos comentado en el punto anterior.

A partir de este momento la inclusión de los componentes en el formulario se corresponde con la inserción de atributos en la clase derivada TForm1. En la figura 1 observamos la interfaz gráfica de los componentes en el formulario y en la figura 2 el correspondiente archivo de cabecera generado automáticamente.

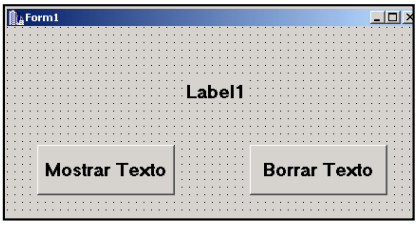

Figura 1. Formulario de la aplicación

Si observamos el código del fichero de cabecera correspondiente al formulario de la figura 1, podemos constatar cómo los tres componentes del formulario son punteros a clases ya existentes, y además, son atributos de la clase derivada a la que pertenece el formulario Form1.

```
   class TForm1 : public TForm
   {
   __published:
       TButton *Button1;
       TButton *Button2;
       TLabel *Label1;
   private:   // User declarations
   public:    // User declarations
        __fastcall
TForm1(TComponent* Owner);
   };
   //---------------------------
   extern TForm1 *Form1;
```

Figura 2. Clase derivada TForm1

El texto que presentan los botones corresponde con el establecimiento de su propiedad *Caption*, ésta propiedad puede establecerse en tiempo de diseño o mediante una instrucción de asignación C++. Por ejemplo, con el componente Button1, como se ve en la figura 2, es un puntero a la clase TButton, y la propiedad Caption, pertenece entre otras a ésta clase. Sólo queda asignarle el valor correspondiente:

```
Button1->Caption="Mostrar Texto";
```

El operador -> accede al objeto apuntado por Button1 y selecciona el atributo deseado, en este caso Caption. Análogamente debería realizarse con Button2.

El segundo paso nos mostrará cómo la escritura de los manejadores o gestores de evento de nuestro programa de ejemplo supone la inclusión de funciones miembro en la clase derivada TForm1. En este caso particular, los eventos que debemos contemplar son la ejecución de un click en cada uno de los botones, el primero de ellos mostrará un mensaje en el componente etiqueta, y el segundo borrará el texto de la etiqueta.

En el entorno de desarrollo debemos seleccionar el evento Click del componente botón Button1 pues queremos darle respuesta cuando se produzca, es decir, queremos escribir un manejador para ese evento. Una vez seleccionado escribimos código C++ con la instrucción correspondiente, la asignación del texto al atributo Caption de la Clase TLabel en este caso. Esta instrucción se realiza de manera similar en Button2.

```
   //------------------------------
   void __fastcall
TForm1::Button1Click(TObject *Sender)
   {
   Label1->Caption = "Jenui 2003";
   }
   //------------------------------
   void __fastcall
TForm1::Button2Click(TObject *Sender)
   {
   Label1->Caption = "";
   }
   //------------------------------
```

Figura 3. Archivo cpp con los manejadores de evento

La clase Tlabel, al igual que la clase TButton, posee entre sus atributos de clase la propiedad Caption. El borrado del texto se realiza en este caso asignando la cadena vacía a la propiedad Caption. Una vez hecho esto, el ejecutable generado espera a que el usuario haga click en alguno de los botones para ejecutar el gestor de

evento correspondiente. La funcionalidad se presenta en la figura 4.

Figura 4. Ejecución de la aplicación de ejemplo

Hasta aquí la realización del programa. Sin embargo, existen aspectos importantes que refuerzan la idea del paralelismo entre el paradigma de la orientación a objetos y el desarrollo de aplicaciones dirigidas por eventos. En la figura 3 se observa cómo las funciones manejadoras de eventos son en realidad funciones miembro de la clase TForm1, pues presentan el operador de resolución de ámbito (::) en la escritura del cuerpo. Además, accedemos al componente Label1 de igual manera a como se acceden a los atributos de una clase, pues como hemos comentado antes, los componentes son atributos de la clase. Este tipo de detalles son en principio transparente al usuario de la herramienta como al usuario final de la aplicación.

Resta comentar otro aspecto importante, sabemos que en C++ los prototipos de las funciones son obligatorios, sin embargo no hemos comentado nada al respecto y la aplicación funciona perfectamente. Esto es debido a que en cada archivo .cpp se incluye el .h correspondiente y el entorno de desarrollo ha incluido automáticamente los prototipos en el fichero de cabecera. ¿De qué manera se han incluido los prototipos? Los prototipos están incluidos como correspondientes funciones miembro de la clase derivada TForm1, así se comprueba en la figura 5.

```
class TForm1 : public TForm
{
__published:
    TButton *Button1;
    TButton *Button2;
    TLabel *Label1;
    void __fastcall
Button1Click(TObject *Sender);
    void __fastcall
Button2Click(TObject *Sender);
  private:   // User declarations
  public:    // User declarations
    __fastcall
TForm1(TComponent* Owner);
  };
  //---------------------------
  extern TForm1 *Form1;
```

Figura 5. Clase TForm1 con funciones miembro

Como se observa en la figura 5, la herramienta incluye los prototipos como métodos de la clase TForm1. El alumno debe conocer este aspecto pues de lo contrario nunca sabrá que está derivando una clase y añadiendo atributos y métodos mientras diseña una aplicación orientada a eventos.

7. Conclusión

Buscamos fomentar el aprendizaje por analogía. En la puesta en práctica de la idea planteada aquí, los alumnos asimilan mucho mejor los conocimientos porque pueden relacionarlos con otras materias recibidas, facilitando su comprensión. Uno puede aprender a desarrollar herramientas dirigidas por eventos ignorando los aspectos de la programación orientada a objetos que se han comentado. Sin embargo, es más sencillo para el alumnado con conocimientos del paradigma de la orientación a objetos saber que en además está derivando clases cada vez que genera un formulario, añadiendo atributos cada vez que selecciona un nuevo componente para ese formulario, y añadiendo métodos cuando escribimos un manejador de evento.

No sólo esto, es obligación del alumnado conocer al detalle la realidad de los programas que desarrolla, y obviar este punto de vista sería acotar la realidad. Una vez tenga suficiente soltura en el desarrollo de este tipo de aplicaciones, el alumno podrá abstraerse de este punto de vista como de cualquier otro, sin embargo para el aprendizaje y los primeros acercamientos a la programación dirigida a eventos, es una gran ayuda y un aspecto fundamental relacionarlo con la orientación a objetos.

Referencias

[1] Aburruzaga García, Gerardo; Median Bulo, Inmaculada & Palomo Lozano, Francisco. *Fundamentos de C++*. Servicio de Publicaciones, Universidad de Cádiz, 2001.

[2] Stroustrup, Bjarne. *El Lenguaje de Programación C++*. Addison-Wesley, 2001.

[3] Wu, C. Thomas. *Introducción a la programación orientada a objetos con Java.* McGraw Hill, 2001.

[4] Charte Ojeda, Francisco. *Programación con C++ Builder.* Anaya Multimedia, 1997

[5] Miano, John; Cabanski, Tom & Howe, Harold. *Borland C++ Builder How-To*. Wait Group Press, 1997.

[6] Calvert, Charles. *Borland C++ Builder Unleashed.* Sams Publishing, 1997.