Capítulo 5

La biblioteca de clases

Desde la llegada del S.O. Windows, las aplicaciones se han desarrollado haciendo uso de la API Win32 (API –Application Programming Interface–) que engloba multitud de funciones relacionadas que se han ido incorporando conforme el S.O. ha ido creciendo. El desarrollo de aplicaciones basadas en esta API, como se hacía en la década de los 90, constituía un trabajo casi artesanal, puesto que los elementos visuales más sencillos debían ser codificados manualmente.

Con idea de evitar el uso de la API de Windows, aparecieron las librerías de elementos prefabricados, junto con otras herramientas de elementos visuales. De esta forma, fabricantes de compiladores –por ejemplo, de C++– desarrollaron paquetes como las MFC (Microsoft Foundation Classes) o OWL (Object Windows Library) de Borland, al mismo tiempo que Microsoft presentaba la primera versión de Visual Basic.

La llegada de Visual Basic significó un cambio radical en la programación como se conocía hasta entonces; sirvió de punto de partida para otros desarrollos, como es el caso de Delphi. Gracias a entornos de tipo RAD (Rapid Application Development), el programador no tiene que acudir directamente a las funciones del API, ya que existen elementos predefinidos que actúan como capa entre el API de Windows y la aplicación.

Con la llegada de Windows95, y el modelo de componentes COM (Component Object Model) tiene lugar una nueva situación aún más favorable; lo que hasta entonces eran grupos de funciones, se convierten en componentes con propiedades y métodos. Además, lo que es más importante, los modelos aportan independencia del lenguaje a nivel binario, ya que puede ser desarrollado –por ejemplo– en C++, y ser usado en Visual Basic. Con la llegada de Windows2000, los servicios proporcionados fueron incrementados hasta constituir COM+, facilitando la adaptación de los componentes mediante atributos.

Desde este punto de vista, .NET es la evolución de COM+; sin embargo, nos "quedamos cortos", ya que .NET es mucho más. La plataforma .NET incorpora una serie de servicios de bajo nivel (la gestión de memoria, seguridad, control de código, etc.), y la biblioteca de clases .NET Framework, con independencia del lenguaje, facilita todas las herramientas para incorporar a las aplicaciones.

Los motivos por los que la plataforma .NET está llamada a ser sobre la que trabajarán los programadores, se basan en que ya no se dispone de funciones y grupos de funciones, sino de clases de objetos que hacen el proceso de codificación mucho más sencillo; así como la independencia del lenguaje y la interoperabilidad.

5.1 El problema de las versiones con .NET Framework

La biblioteca de clases .NET Framework proporcionada por Microsoft, está disponible actualmente en dos versiones: la 1.1 y la 2.0. Lo normal será pensar que la versión posterior engloba a la anterior (es decir que la 2.0 incluye la 1.1), pero no es así exactamente. Microsoft, motivado por el cambio radical en la tecnología y con vista al software futuro, ha desarrollado la versión 2.0 desde el principio, sin partir de la base 1.1. Esto, que a priori parece ser una ventaja a todas luces, puede convertirse en un grave inconveniente. De hecho, para la aplicación desarrollada así es.

La solución no es directa; pero puede evitarse complicaciones siguiendo una de las conductas definidas a continuación:

- Si desea usar la aplicación *Fretting Fatigue* en Windows y no requiere de otros servicios de la biblioteca .NET Framework, se recomienda que haga uso de la versión 1.1 y se olvide de estos problemas.

- Si por el contrario, desea desarrollar otro software .NET en su propio ordenador al tiempo que hace uso de la aplicación, puede tomar dos decisiones:

    - Usar Microsoft Visual Studio.NET 2003, que proporciona la versión 1.1 y puede utilizar la aplicación sin problemas.

    - Usar la nueva aplicación Microsoft Visual Studio.NET 2005 (que de momento está en fase Beta), que proporciona la versión 2.0; en este caso, la opción más acertada sería volver a compilar el código fuente de la aplicación, y asignar una nueva versión a la misma (por ejemplo la 2.0)

5.2 Las librerías

Las librerías son elementos de programación muy eficaces y que permiten redistribuir los recursos de una aplicación de forma que no se carguen todos en la memoria de golpe al ejecutarla, sólo si se requiere su uso.

El formato más extendido es el DLL (bibliotecas de vínculos dinámicos); hoy día puede encontrarse en cualquier aplicación, incluso las de pequeño tamaño.

La biblioteca de clases .NET Framework, proporciona un conjunto de librerías que contienen la información necesaria para la ejecución de las aplicaciones desarrolladas en .NET (la .NET Framework fue definida en el documento Manual de Usuario); sin ellas, la aplicación no puede ejecutar o puede que lo haga pero de una forma inestable y peligrosa para la integridad del sistema.

5.2.1 La ventaja de la actualización

Con la idea de que las librerías proporcionan código precompilado del que puede hacer uso una aplicación en tiempo de ejecución, se prevé otra ventaja: la actualización de

recursos. Piense que si disponemos de un recurso externo a la aplicación (una librería) podemos modificar dicho recurso sin hacer lo mismo con la aplicación.

En la figura 5.1 se muestra esta idea con un ejemplo: una ecuación diferencial. La aplicación genera una ecuación diferencial (piense en una ley de crecimiento, por ejemplo) que debe ser resuelta de la mejor forma posible; se dispone de una librería de funciones que permite realizar dicha tarea, devolviendo a la aplicación el resultado.

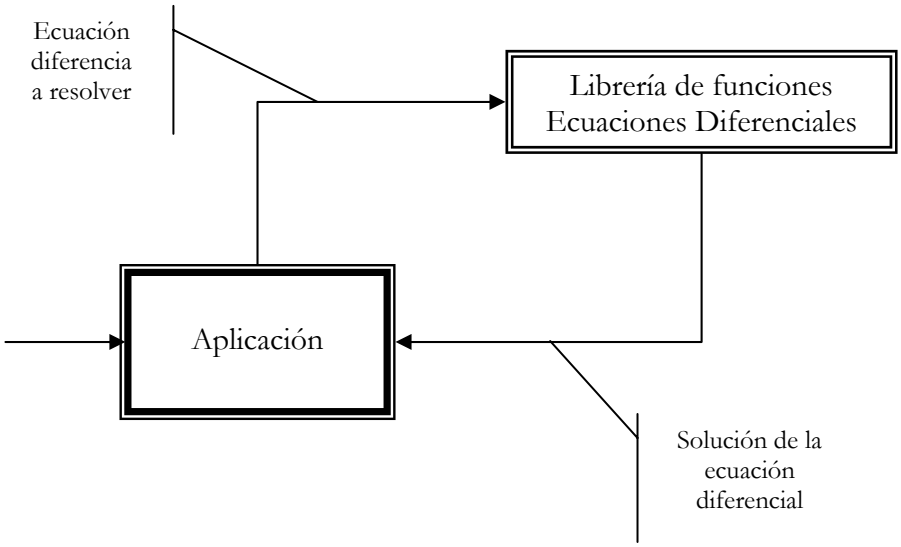

Figura 5.1: Librería de ecuaciones diferenciales. Ejemplo.

Suponga que la aplicación no resulta válida para un grupo de datos de entrada (piense que se trata de un grupo de ensayos de fretting, con material con comportamiento diferente a los estudiados, por ejemplo); la pregunta es: ¿debería deshacerse de la aplicación y estudiar estos casos por separado? Indudablemente, si la aplicación engloba todas las funciones la respuesta es sí. Pero la ventaja de externalizar las funciones se hace patente: puede modificar la librería para que en determinadas situaciones haga uso de nuevas funciones (que pueden añadirse a las existentes en la librería o en otra independiente)[2].

5.2.2 Las librerías en la comunidad

Según esto, puede desacoplarse el estudio de los problemas particulares y la implementación de métodos de su resolución; es el caso del ejemplo, la resolución de ecuaciones diferenciales, pero también podrá encontrar librerías de resolución de integrales, de métodos de derivación, de ajuste de curvas, etc. Las posibilidades que admite son bastante amplias.

Gracias a ello –y al auge de Internet– pueden localizarse librerías realizadas por programadores de todo el mundo (ventaja de la ciencia, la internacionalidad), de los casos más variados: representaciones gráficas, mejora de programas comerciales, etc. Claro que

[2] La solución para el caso de no disponer de librerías externas, pasa primero por la externalización de las mismas, para proseguir con la solución propuesta en este ejemplo.

una librería es una "caja negra", se conocen los datos de entrada y que proporciona unos datos de salida (en determinadas situaciones puede probarse su bondad, pero no siempre es el caso), los paso intermedios son desconocidos salvo para el creador de la librería; incluso las posibilidades que permite son desconocidas.

Por ello, las librerías suelen estar acompañadas de lo que se llaman ficheros cabecera, que contienen la declaración de las funciones y una breve descripción de su funcionalidad.

En el caso de la OOP, las librerías cobran aún más importancias, ya que la implementación de clases es una tarea muy delicada que requiere cierto grado de soltura con el lenguaje y con los fundamentos de la OOP.

5.3 Las librerías de la .NET Framework

La biblioteca de clases de .NET está compuesta por un conjunto de ensamblados formados a partir de la unión de módulos. Estos módulos representan las librerías DLL y archivos ejecutables.

Las aplicaciones comparten todos estos ensamblados, para ello se alojan en un "punto" del sistema de archivos cocido como el GAC (Global Assembly Cache) al que tienen acceso. Puede comprobarse en el Panel de Control de Windows a través de la utilidad que se crea al instalar la biblioteca de clases .NET Framework (ver figura 5.2)

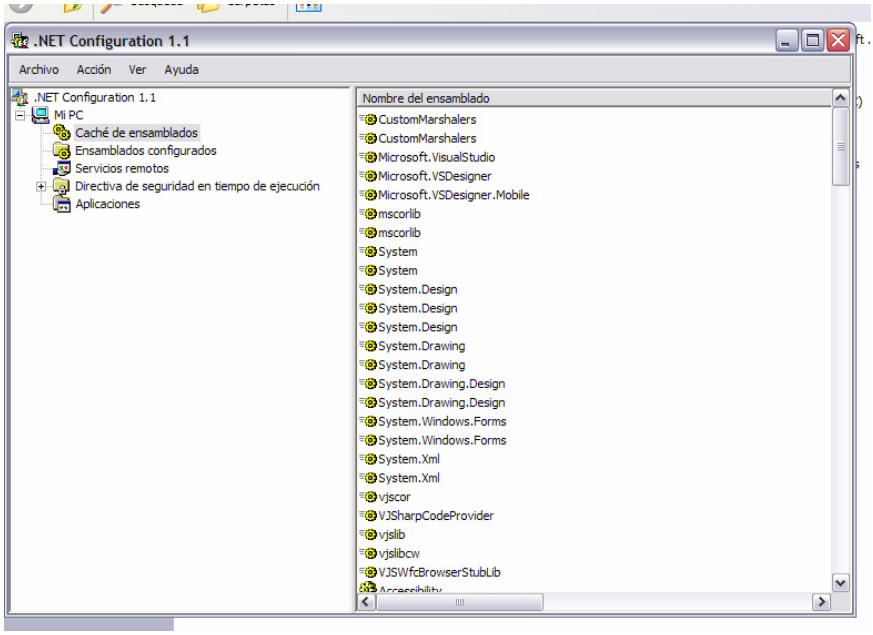

Figura 5.2: Vista del Caché de Ensamblados de Windows.

5.3.1 Clases fundamentales

Las clases que pueden encontrarse en la .NET Framework pueden agruparse en varios niveles:

- las clases comunes a todos los lenguajes (clases básicas)
- las clases genéricas que hacen uso de las anteriores
- las clases específicas según cada tipo de aplicación

Todas las aplicaciones desarrolladas tienen un ensamblado común (*mscorlib*), que contiene las clases básicas, indispensables para el funcionamiento de cualquier aplicación (sin importar el tipo). Forman parte de éste:

- las definiciones de tipos que constituyen el CTS (Common Type System), un sistema de tipos de datos común a todos los sistemas .NET (Byte, String, Double…)[3]
- los servicios de seguridad, control de la ejecución de múltiples hilos y las comunicaciones entre sistemas remotos, entre otros;
- así como de los aspectos de bajo nivel: la comunicación con los compiladores o la recuperación de información durante la ejecución

[3] La existencia de un sistema común de tipos hace posible que la biblioteca de clases de .NET pueda ser utilizada desde cualquier lenguaje de programación, en contraste con otro tipo de bibliotecas que son de uso exclusivo de un lenguaje.

Además de las que pueden encontrarse en *mscrolib*, existen otra serie de ensamblados, los correspondientes a las clases genéricas o de uso general. Se trata de servicios de alto nivel accesibles para todas las aplicaciones .NET independientemente de su tipo. También forma parte de esta categoría el ensamblado encargado de los servicios de diagnósticos, configuración y comunicación a través de redes (*System.dll*).

Sobre las clases básicas y las genéricas se encuentran las específicas para cada tipo de aplicación. No es lógico que los servicios requeridos por Aplicación de Consola se ubiquen en el mismo ensamblado que el de las Aplicaciones Web, por tanto ocupan ensamblados distintos

Esquema de bloques

Todos los ensamblados de la .NET Framework forman la biblioteca de clases disponibles para cualquier aplicación .NET. En el esquema de bloques de la figura 5.3 se muestra de forma gráfica la dependencia de cada una de las librerías.

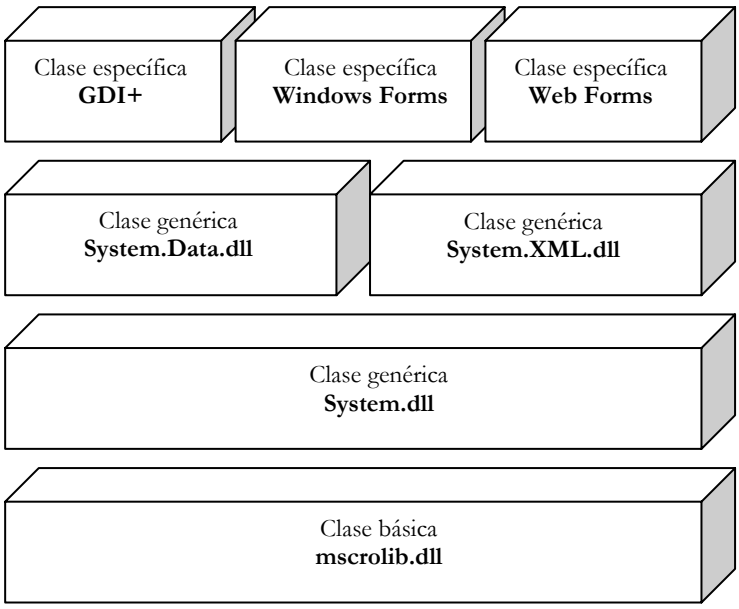

Figura 5.3: Esquema de bloques de la .NET Framework.

Contenido de un ensamblado .NET

Los elementos que componen un ensamblado pueden clasificarse en cuatro grupos principales:

- datos sobre el ensamblado: dependencias, atributos, etc.
- datos de tipo
- código MSIL (Microsoft Intermediate Language); y,

- recursos

5.3.2 Las referencias a librerías

Al iniciar una aplicación en un entorno de desarrollo como Visual Studio.NET, de forma automática se incluyen las referencias necesarias a cada uno de los ensamblados comentados anteriormente; si bien, para hacer uso de otras librerías que no sean propias de la .NET Framework (como la del ejemplo de la ecuación diferencial), debe añadirse su referencia de forma manual.

En particular, en Visual Studio.NET debe añadirse sobre el Explorador de soluciones, con la opción "Agregar referencia", seleccionando la librería que quiera usarse. El uso de los recursos proporcionado por ésta es idéntico al de la .NET Framework.

5.3.3 El código intermedio

Cuando se compila (desde Visual Studio.NET o desde cualquier otro entorno que use el compilador .NET) se genera información de dos tipos: metadatos y código objeto.

El código objeto no es un código nativo para cualquier procesador, sino un código independiente de cualquier plataforma (de hardware y software). Se le conoce como código intermedio (MSIL –Microsoft Intermediate Language–) y está diseñado para ser compacto y transportable entre dispositivos. Todos los lenguajes .NET generan este mismo código; esto, unido a la existencia de un sistema de tipos comunes y al uso común de las librerías de .NET, hace posible la interoperabilidad entre lenguajes.

Las ventajas de esta forma de proceder no se limitan a la interoperabilidad, sino que también hace posible que una misma aplicación, desarrollada y compilada una única vez, puede ser ejecutada satisfactoriamente en cualquier sistema operativo.

5.3.4 La ejecución

Completado todo el proceso anterior de desarrollo y compilación de la aplicación sobre un S.O., debe ejecutarse la aplicación para que comience a realizar su función. El código MSIL se carga en la memoria de forma supervisada y debe ser compilado a código nativo para que funcione; de eso se encarga el entorno CLR (Common Language Runtime).

El CLR se encarga de abrir los ensamblados y preparar para ellos un espacio en memoria, provee de todos los servicios fundamentales de la plataforma .NET; haciendo uso del compilador JIT (Just in-time).

Esquema general

En la figura 5.4 se muestra el esquema de desarrollo y ejecución en la plataforma .NET.

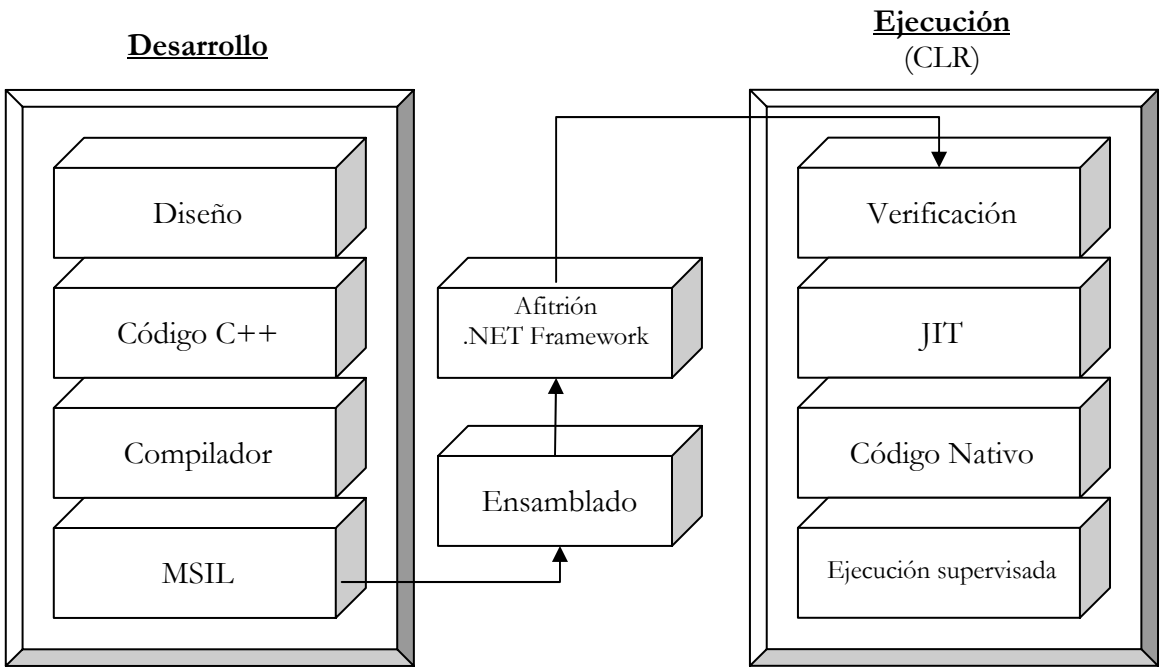

Figura 5.4: Esquema general de ejecución de aplicaciones .NET