Diez cosas que debe saber sobre IVI-COM

Por Stefan Kopp, Agilent Technologies

Stefan Kopp es licenciado en informática por la Universidad de Educación Cooperativa de Stuttgart, Alemania. Trabaja para Agilent (antes HP) desde 1989, donde ha ocupado distintos puestos, incluidos el de jefe de proyectos para soluciones de prueba personalizadas y el de representante de ventas. Actualmente ejerce de consultor en el área de componentes de sistemas de prueba y automatización de pruebas.

IVI-COM es la norma más reciente para controladores de instrumentos. En los últimos tiempos está recibiendo más atención debido a que el Consorcio LXI la recomienda. Hay quienes dicen que el software COM es poco práctico y difícil de usar. Es difícil apreciar lo que no se conoce... Quienes estén dispuestos a comprobarlo por sí mismos deberán saber diez cosas sobre IVI-COM. Lea bajo su responsabilidad, ya que es posible que termine gustándole esta tecnología.

Compatibilidad total

Microsoft introdujo el Modelo de objetos componentes (COM) en 1995 con el fin de solucionar una serie de problemas que se habían detectado en las DLL. Uno de ellos es que, a menudo, hay grandes diferencias entre las DLL. Se dejan al criterio del programador individual o del entorno de programación aspectos como el tipo de datos, los nombres de funciones, las convenciones de llamada y la gestión de errores.

Este grado de flexibilidad puede convertirse en un verdadero quebradero de cabeza para los diseñadores de aplicaciones que necesitan que los módulos de software se comuniquen entre sí. Pueden verse obligados a adaptar el código de las aplicaciones a cada DLL individual, especialmente si se han escrito en entornos de programación distintos. Como último recurso, tendrían que crear *glueware* adicional para que los módulos pudieran comunicarse.

COM cambia la ecuación eliminando cierta flexibilidad: incluye una definición de tipos de datos independiente del lenguaje, nombrado y convención de llamada, gestión de errores y una norma de documentación.

COM se popularizó a finales de los años noventa. Además, es la tecnología en la que se basa OLE/ ActiveX. Gracias a esta popularidad, todos los entornos de programación disponibles en la actualidad soportan COM. En otras palabras, los controladores basados en COM son compatibles con todos los lenguajes de programación modernos, lo que supone una compatibilidad total.

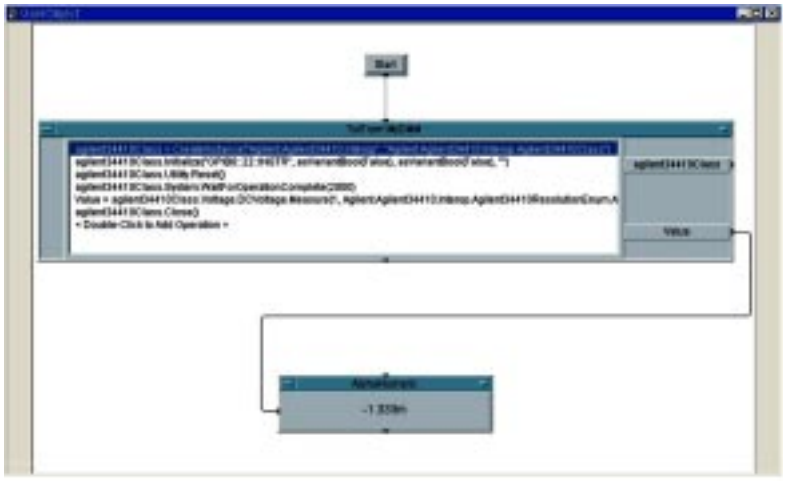
Si dispone de un controlador de instrumentos para *el* lenguaje específico elegido, ¿por qué habría de interesarle hacerlo compatible con *todos* los lenguajes?

Los fabricantes de instrumentación sólo pueden dedicar una determinada cantidad de esfuerzo y energía para el desarrollo de controladores de instrumentos. La posibilidad de concentrarse en un único controlador permitirá que éste sea más completo y de mayor calidad.

Veamos cómo funciona en la práctica la independencia de lenguajes. El código Visual BASIC siguiente muestra cómo realizar una medida básica de tensión DC empleando el controlador IVI-COM para el multímetro digital Agilent 34410A.

```
Dim MyDMM As Agilent34410 ' Declare object variable
MyDMM = New Agilent34410 ' Create object
MyDMM.Initialize("GPIB0::22::INSTR", False, False) ' Initialize instrument session
MyDMM.Utility.Reset() ' Reset instrument
MyDMM.System.WaitForOperationComplete(2000) ' Wait for completion
' Do a DC voltage measurement
Dim Result As Double
Result = MyDMM.Voltage.DCVoltage.Measure(1.0,
  Agilent34410ResolutionEnum.Agilent34410ResolutionDefault)
' Display result
MsgBox("Value: " & Result)
' Close instrument session
MyDMM.Close()
```

Como es lógico, esta misma funcionalidad es muy similar cuando se implementa en Agilent VEE:



También es posible llamar a los controladores IVI-COM desde NI LabVIEW y LabWindows/CVI, un entorno ANSI-C (no orientado a objetos). Éste es el ejemplo para CVI:

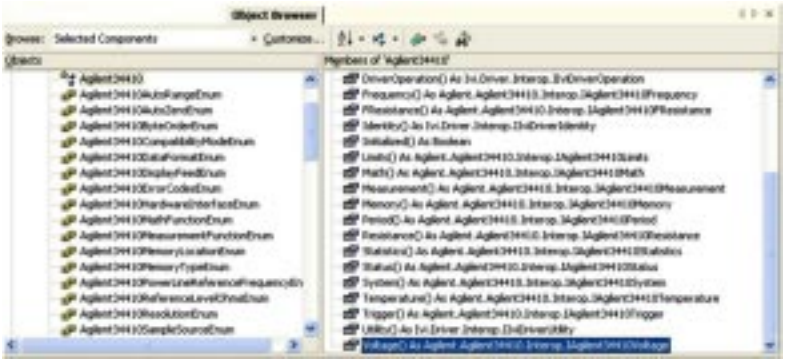
```
static CAObjHandle MyDMM,MyDMMUtility,MyDMMSystem,MyDMMVoltage,MyDMMDCVoltage;
double Result;
char s[100];
// Create driver object
Agilent34410Lib_NewIAgilent34410 (NULL, 1, LOCALE_NEUTRAL, 0, &MyDMM);
// Read interface pointers
Agilent34410Lib_IAgilent34410GetUtility (MyDMM, NULL, &MyDMMUtility);
Agilent34410Lib_IAgilent34410GetSystem (MyDMM, NULL, &MyDMMSystem);
Agilent34410Lib_IAgilent34410GetVoltage (MyDMM, NULL, &MyDMMVoltage);
Agilent34410Lib_IAgilent34410VoltageGetDCVoltage (MyDMMVoltage, NULL, &MyDMMDCVoltage);
// Initialize instrument session
Agilent34410Lib_IAgilent34410Initialize (MyDMM, NULL, "GPIB0::22::INSTR", VFALSE, VFALSE, "");
// Reset instrument
Agilent34410Lib_IIviDriverUtilityReset (MyDMMUtility, NULL);
// Wait for completion
Agilent34410Lib_IAgilent34410SystemWaitForOperationComplete (MyDMMSystem, NULL, 2000);
// Do a DC voltage measurement
Agilent34410Lib_IAgilent34410DCVoltageMeasure
  (MyDMMDCVoltage, NULL, 1.0,Agilent34410LibConst_Agilent34410ResolutionDefault,&Result);
sprintfs,"%F",Result);
MessagePopup ("Result", s);
// Close instrument session
Agilent34410Lib_IAgilent34410Close (MyDMM, NULL);
```

En todos los ejemplos de programación de este artículo se ha omitido la gestión de errores por motivos de claridad.

Orientación a objetos

En los años noventa se popularizó la programación orientada a objetos (OOP). En la OOP, la funcionalidad de un módulo de software se organiza en torno a objetos del "mundo real" con los que trata la aplicación (por ejemplo, pedidos, empleados, impresoras o multímetros). Los objetos contienen variables (que describen el estado del objeto y otros atributos relevantes) y funciones que manipulan los datos. En la OOP, funciones y variables se denominan *métodos* y *propiedades*, respectivamente. La OOP ofrece una gran potencia gracias a conceptos como *herencia de objetos*, mediante la cual se obtiene una nueva clase de objetos a partir de una existente y se heredan los métodos y las propiedades de esa clase. Si es necesario, la nueva clase puede anular los métodos seleccionados con una implementación personalizada. Todos los lenguajes de programación modernos soportan la programación orientada a objetos. Existe un consenso generalizado de que el software orientado a objetos es más fácil de modificar y mantener que el software tradicional (en particular si interviene un equipo de programadores).

Por su diseño, todo el software COM está orientado a objetos. Los entornos de programación modernos están pensados para la OOP: gestionan los controladores IVI-COM como si fuera una clase "nativa" implementada en su entorno. Por tanto, es posible utilizar las ayudas y herramientas de programación incorporadas cuando se trabaja con el controlador. Por ejemplo, Visual Studio permite inspeccionar el controlador utilizando su *Examinador de objetos* (véase a continuación).



Controlador 34410A IVI-COM, tal como se muestra en el Examinador de objetos de Visual Studio

¿Debe diseñar su aplicación orientada a objetos para utilizar controladores IVI-COM? Naturalmente que no. Puede adoptar la metodología de procedimientos clásica de la programación y seguir disfrutando de las ventajas de IVI-COM.

Estructura

Muchos tipos de controladores de instrumentos se basan en DLL y contienen una lista desordenada de funciones (por ej., VXIplug&play[1]). No hay una estructura inherente en el controlador que ayude a identificar la función que se necesita y los nombres son a menudo poco descriptivos debido a su escasa longitud.

COM utiliza *interfaces* para agrupar la funcionalidad relacionada en un módulo. Por ejemplo, en un controlador IVI-COM, todos los métodos y propiedades relacionados con el disparo se encontrarían probablemente en una única interfaz, llamada ITrigger (por convención, los nombres de interfaz van precedidos de una "I"). Esta agrupación lógica permite familiarizarse fácilmente con el controlador. Los controladores IVI-COM van un paso más allá. Organizan las interfaces en una estructura de árbol o de red (véase a continuación).

[1] Los paneles de funciones que forman parte de VXIplug&play ofrecen una vista en árbol de las funciones que contiene el controlador DLL. Sin embargo, los paneles de funciones son un complemento propietario "artificial" de la DLL del controlador, definida por la Alianza VXIplug&play. Sólo los lenguajes de medida como Agilent VEE son capaces de interpretar estos archivos.

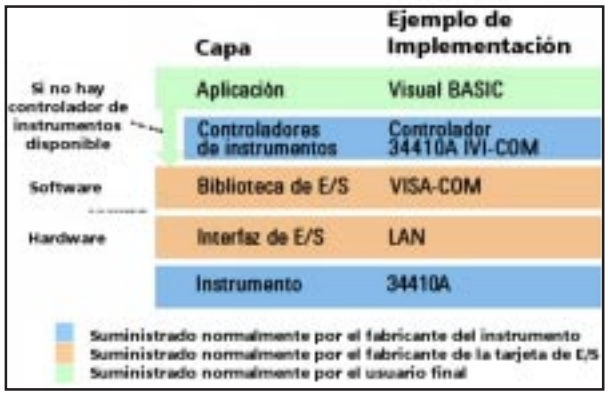
Estructura de árbol del controlador 34401A IVI-COM (documentado en el archivo de ayuda del controlador)

Así es como funciona. Inicialmente, cuando se "crea" el objeto del controlador, se devuelve un puntero a la *interfaz predeterminada* (en este ejemplo, IAgilent34401). Esta interfaz de nivel superior suele contener unos pocos métodos elementales y muchas de las llamadas propiedades de puntero, que hacen referencia a otras interfaces. Al leer una propiedad de puntero y "des-referenciar" el puntero, podrá acceder a los métodos y las propiedades de esa otra interfaz.

En Visual BASIC, VEE y otros lenguajes, puede navegar cómodamente por la red de interfaces y métodos del controlador utilizando el operador punto, por ejemplo, DMM.Utility.Reset(). Véanse también los códigos de ejemplo anteriores.

Independencia de la interfaz de E/S

Al igual que los controladores VXIplug&play, los controladores IVI se basan en la biblioteca VISA E/S (véase a continuación).



Las cinco capas de las aplicaciones de prueba

Hasta cierto punto, VISA no es más que un detalle de implementación, oculto a los ojos del programador de aplicaciones (que llama a las funciones de controlador). No obstante, VISA es un elemento clave, ya que proporciona una interfaz de programación coherente para distintas interfaces de E/S, incluidas LAN, USB, GPIB, VXI y RS232.

Muchos instrumentos modernos incorporan interfaces de E/S alternativas (por buenas razones, por ejemplo, USB es ideal para la automatización de pruebas específicas en laboratorio, mientras que LAN suele ser la mejor opción para sistemas de prueba). A través de VISA, el soporte para estas distintas interfaces en el controlador de instrumentos se suministra prácticamente gratis. Para utilizar otra interfaz, lo único que debe hacer es suministrar una cadena de dirección VISA distinta al método *Initialize* del controlador. La tabla siguiente muestra algunos ejemplos de cadenas de recurso VISA.

Cadena de dirección VISA	Descripción
GPIB0::17::INSTR	Instrumento GPIB en la dirección 17
TCPIP0::134.40.173.138::INSTR	Instrumento LAN / protocolo "VXI-11"
TCPIP0::134.40.173.138::SOCKET	Conexión instrumento LAN / socket TCP

Intercambiabilidad y portabilidad

Uno de los objetivos de IVI es conseguir un mayor grado de normalización en la capa del controlador de instrumentos. La norma VXIplug&play, predecesora de IVI, presentaba un grado moderado de normalización en esta capa (especificando aspectos de bajo nivel como tipos de datos y gestión de errores). Sin embargo, IVI va un paso más allá y especifica los prototipos de función exactos (nombres y parámetros de función) que debe implementar el controlador. Naturalmente, las funciones y los parámetros dependen de la capacidad del instrumento. Por consiguiente, la Fundación IVI definió varias clases de instrumentos, por ejemplo, para fuentes de alimentación, osciloscopios, conmutadores, multímetros digitales y analizadores de espectros.

La normalización de las funciones de controlador en cada clase permite aislar el software de la aplicación de los detalles del instrumento individual. En consecuencia, es posible cambiar un multímetro digital por otro sin modificar el código de la aplicación. De forma semejante, el mismo software de aplicación puede ejecutarse en varios sistemas de prueba aunque la configuración de su hardware sea algo diferente.

Es necesario reconocer, sin embargo, la dificultad (incluso la imposibilidad) de conseguir una *completa* intercambiabilidad del hardware sin realizar ninguna modificación en el software. Esto se debe a que muchos instrumentos presentan características únicas a las que sólo se pueden acceder a través de funciones no estándar. Sin embargo, IVI puede ayudar a minimizar los cambios en el software debidos a cambios en el hardware.

El código de ejemplo siguiente es similar a los anteriores. Sin embargo, en este caso se utilizan métodos compatibles con clases IVI. Por tanto, este código podría utilizarse sin modificaciones con otros multímetros digitales (siempre que incluyan un controlador IVI-COM).

```
Dim MyDMM As New Agilent34410 ' Declare object variable and create object
Dim MyIVIDMM As IviDmm ' Declare variable for IVI DMM class interface
MyIVIDMM = MyDMM ' Retrieve IVI DMM class interface for MyDMM object
' Now use methods in IVI DMM class interface
MyIVIDMM.Initialize("GPIB0::22::INSTR", False, False)
MyIVIDMM.Utility.Reset()
MyIVIDMM.Function = IviDmmFunctionEnum.IviDmmFunctionDCVolts
Dim Result As Double
Result = MyIVIDMM.Measurement.Read(2000)
MsgBox("Value: " & Result)
' Close instrument session
MyIVIDMM.Close()
```

El único elemento específico del instrumento en el código anterior es la definición y creación de la clase de controlador, Agilent34410. Si desea un código verdaderamente portable, no podrá utilizar directamente el nombre de clase. En su lugar, utilice IVI Session Factory. Este servicio aceptará un nombre simbólico (portable), recuperará la clase de controlador correspondiente de una base de datos de configuración y creará el objeto del controlador.

```
Dim Factory As New IviSessionFactory ' Create object for IVI Session Factory
Dim MyIVIDMM As IIviDmm ' Declare variable for IVI DMM class interface
Set MyIVIDMM = Factory.CreateDriver("MyDMM") ' Let session factory create driver
object
MyIVIDMM.Initialize("MyDMM", False, False) ' Use symbolic name instead of address
string
MyIVIDMM.Utility.Reset()
MyIVIDMM.Function = IviDmmFunctionEnum.IviDmmFunctionDCVolts
Dim Result As Double
Result = MyIVIDMM.Measurement.Read(2000)
MsgBox("Value: " & Result)
' Close instrument session
MyIVIDMM.Close()
```

Rendimiento

Hay varias versiones de COM. Una de ellas, llamada Distributed COM (DCOM), funciona a través de Ethernet. Se basa en llamadas a procedimientos remotos (RPC) y es relativamente lenta debido a la carga adicional que supone el envío de parámetros y la devolución de valores a través de la red. Sin embargo, *no* es ésta la versión de COM que utiliza IVI-COM. IVI-COM se basa en la versión DLL de COM. Como ya hemos señalado, la DLL debe cumplir la convención de llamada, los tipos de datos, la gestión de errores, etc. que especifica COM. Sin embargo, llamar a un método COM dentro de una DLL es tan rápido como llamar a cualquier otra función DLL. Las DLL se cargan en el espacio de dirección de la aplicación que realiza la llamada y acceden a los trabajos de forma eficiente a través de llamadas de función.

Flexibilidad

En condiciones ideales, los controladores de instrumentos cubren el 100% de la funcionalidad del instrumento. Algunas veces, sin embargo, cubren un porcentaje menor, un subconjunto que se considera adecuado para el uso automatizado del instrumento. ¿Qué puede hacer en las raras situaciones en las que la capacidad que usted requiere no está disponible en el controlador? La mayoría de los controladores IVI-COM cuentan con métodos de paso a través SCPI que permiten enviar comandos nativos de los instrumentos de forma ordenada a través del controlador.

El ejemplo siguiente utiliza comandos SCPI enviados a través del controlador de instrumentos para recuperar la cadena de ID del instrumento.

```
Dim MyDMM As New Agilent34410 ' Declare object variable and create object
MyDMM.Initialize("GPIB0::22::INSTR", False, False) ' Initialize instrument session
MyDMM.Utility.Reset() ' Reset the instrument
MyDMM.System.WaitForOperationComplete(2000) ' Wait for completion
MyDMM.System.IO.WriteString("*IDN?") ' Ask for ID string using SCPI *IDN? command
Dim s As String
s = MyDMM.System.IO.ReadString ' Read instrument's response
MsgBox("ID String: " & s)
' Close instrument session
MyDMM.Close() ' Close instrument session
```

Versiones

Las DLL clásicas presentan otro problema. A menudo se comparten entre varias aplicaciones. Una aplicación nueva instalada en el sistema puede incluir una nueva versión de una DLL compartida que incluya funcionalidad mejorada o añadida. Pequeños cambios en el comportamiento de la DLL actualizada pueden hacer que la aplicación antigua deje de funcionar correctamente. Las aplicaciones pueden hacerse incompatibles simplemente porque requieran versiones distintas de la misma DLL compartida. Estos problemas y otros similares han pasado a conocerse como *DLL Hell (El infierno de las DLL)*.

COM evita este problema permitiendo que coexistan varias versiones de la misma funcionalidad en un mismo módulo. El software antiguo sigue utilizando la funcionalidad existente sin modificar. Al mismo tiempo, el nuevo software puede aprovechar una versión actualizada de la funcionalidad.

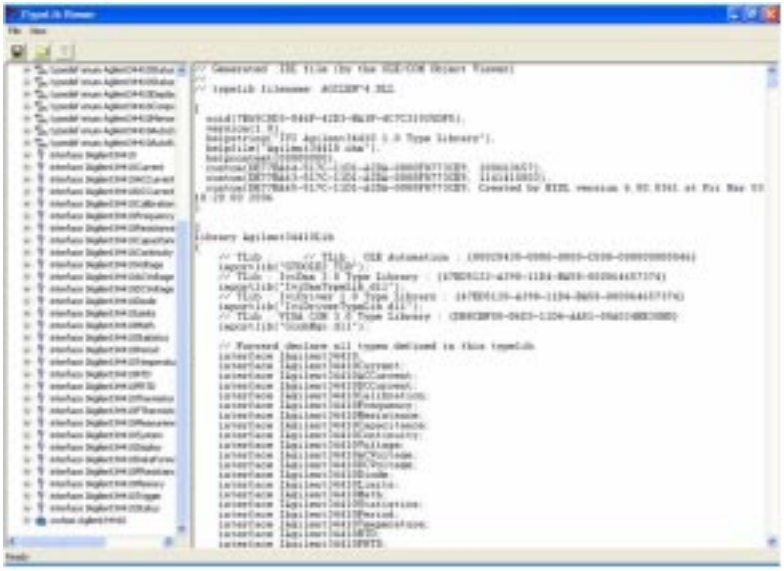
Documentación

Los módulos COM incluyen una descripción de su contenido (clases, interfaces, métodos, propiedades) con la forma de una *biblioteca de tipos*, bien oculta en el archivo de controlador principal o como archivo de biblioteca de tipos aparte (*.TLB o *.OLB). Esta documentación electrónica contiene toda la información necesaria para crear objetos, métodos de llamada, etc.

El entorno de programación lee la información sobre la biblioteca de tipos, que le permite facilitar el acceso al módulo de software de forma sencilla y elegante (véase también Facilidad de uso)

Antes de la introducción de COM se utilizaban archivos de cabecera para especificar las funciones de un controlador, así como sus parámetros. Sin embargo, estos archivos de cabecera dependen del lenguaje

de programación y no contienen tanta información como una biblioteca de tipos COM. Las bibliotecas de tipos tienen formato binario, pero si desea inspeccionarlas por curiosidad, Microsoft le ofrece una herramienta, llamada *Visor de objetos OLE y COM*, que le permitirá hacerlo. Se incluye con Visual Studio (ejecute oleview.exe). Por ejemplo, éste es el controlador 34410A IVI-COM tal como se muestra en el Visor de objetos OLE y COM:



Generalmente no es necesario inspeccionar la biblioteca de tipos: los controladores IVI-COM también incluyen un archivo de ayuda de Windows para el usuario.

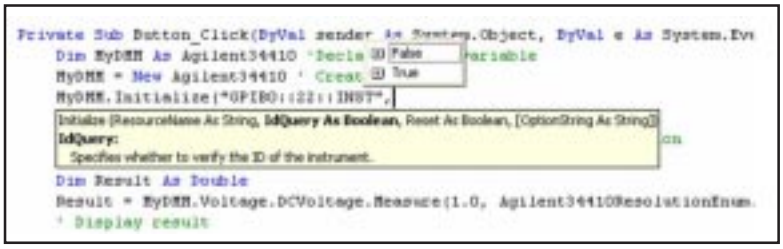
Facilidad de uso

Las aplicaciones que utilizan DLL "ordinarias" pueden ser frágiles (son muy sensibles a los cambios en la ubicación de las DLL). Generalmente, la ubicación debe formar parte de la ruta de búsqueda de Windows para que el sistema operativo pueda buscar y cargar los módulos. Los cambios en el directorio actual o en las variables de la ruta de búsqueda de Windows pueden ocasionar problemas.

A diferencia de las DLL ordinarias, los módulos COM se suelen anotar en el registro del sistema durante las rutinas de instalación. Las aplicaciones y el propio Windows pueden buscar en el registro detalles sobre la instalación de un determinado módulo. Por ejemplo, cuando una aplicación pide a Windows que cargue un módulo COM, Windows podrá recuperar la ubicación exacta de la DLL del controlador a través del registro.

La facilidad de uso resulta aparente cuando se hace referencia a un controlador (en otras palabras, cuando se pide al entorno de programación que cargue y resuma el contenido de una biblioteca de tipos). El entorno de programación podrá compilar una lista de controladores instalados basándose en las entradas del registro del sistema. Seguidamente podrá seleccionar la biblioteca adecuada basándose en el nombre descriptivo del módulo. El ejemplo siguiente muestra una lista de módulos COM que ofrece Visual Studio:

Otra función útil de Visual Studio es *Intellisense* (Predicción de código y Autocompletado de código). Esta función resulta especialmente potente con controladores IVI-COM porque gestiona punteros de interfaz y, por tanto, permite navegar por la red de métodos del controlador (véase el ejemplo siguiente).



Microsoft Intellisense (Code Insight y Code Completion) en funcionamiento

Resumen

IVI-COM presenta claramente una serie de ventajas sobre las normas de controladores de instrumentos tradicionales. La más importante con diferencia es que estos controladores funcionan, literalmente, con todos los lenguajes de programación modernos.

La mayoría de los lenguajes gestionan el controlador como una clase nativa y la integración es perfecta. Por ello, una vez que se tiene práctica con estos controladores, son muy efectivos y fáciles de usar. ❑