Framework para integrar el acceso a dispositivos

Framework for the integration with external

Adonis Cesar Legón Campo

Universidad de las Ciencias Informáticas (UCI)

alegon@uci.cu

Resumen

En este trabajo se describe el análisis, diseño e implementación de un *framework* que permite la integración de aplicaciones con dispositivos externos, partiendo de la necesidad de muchos sistemas de poder interactuar con dispositivos para realizar procesos de capturas de datos e imágenes, lectura y comprobación de documentos, acceso a tarjetas inteligentes, entre otros. Este *framework* propone una metodología y un esquema que simplifica el desarrollo a los distintos módulos o aplicaciones de cualquier sistema, para realizar una interacción sencilla y de alto nivel de programación con los dispositivos que se deben utilizar y extensible para usar otros equipos, luego de haber realizado el estudio de las características y funcionalidades necesarias de cada uno de estos, con el objetivo de cumplir con los requisitos definidos en los sistemas que lo necesiten. Su aplicación siempre puede ser llevada a cabo en cualquier sistema que tenga características semejantes a las tecnologías usadas para esta implementación o incluso en otros sistemas, debido a que su diseño es independiente de la plataforma que se esté utilizando.

Palabras clave: dispositivo externo, extensible, *framework*, independiente de la plataforma, integración.

Abstract

This work describes the analysis, design and implementation, of a framework that allows applications to integrate with external devices, beginning with the need for the systems to interact with devices in order to make data and images capture, documents reading and verification, accessing smartcards, among others. This framework proposes a methodology and scheme, that simplifies the development of the modules or applications of any system, to make a simple and very high programming level integration, with the devices that are going to be used and even extensible to use other devices, after having made a study of the needed characteristics and functionalities for each one of them, so that the systems that are going to use it, can fulfill with the requirements defined. It can be applied in any system with similar technologies as the ones used in this implementation or even in other systems , because it was design in a platform independent way.

Keywords*:* extensible, external device, framework, integration, platform independent.

Introducción

Como parte de los procesos que se desarrollan en muchos sistemas y aplicaciones, es necesaria la captura de datos a través de distintos dispositivos externos especializados en cada tipo de captación, los cuales, luego de ser analizadas las distintas variantes presentes en el mercado a partir de las características y las funcionalidades que brindan, haciéndolas coincidir con los requerimientos para los que serán destinados, comienza un proceso de asimilación en

el cual se persigue el objetivo de lograr una perfecta integración de sus funcionalidades principales con las aplicaciones del sistema.

Debido a las características de los distintos negocios, al requerir el uso de dispositivos externos, surge la necesidad de crear un *framework* para la interacción con dispositivos, de forma sencilla y transparente a la implementación de los negocios, a través de interfaces que describan su comportamiento. Para el diseño de este *framework* de acceso a dispositivos se debe cumplir un conjunto de características fundamentales, que parten del análisis de las principales necesidades de los sistemas y aplicaciones que lo puedan usar:

Integrable: se debe integrar de forma fácil y segura a cualquier aplicación o módulo, de forma tal que pueda ser usado en los distintos procesos de captación definidos en un sistema.

Extensible: es necesario que el *framework* permita la incorporación de nuevos dispositivos en la medida que vayan surgiendo las necesidades para nuevos módulos, o la aceptación de otros equipos con las mismas características que el actual pero de otros fabricantes, de forma tal que se puedan hacer reemplazos sin hacer cambios en las aplicaciones.

Componentes Gráficos: algunos tipos de dispositivo de captura, requieren de retroalimentación gráfica en la interfaz de la aplicación, para lo cual se debe proveer de los componentes gráficos necesarios para su fácil integración, y de esa forma cumplir con los requisitos de crear una interfaz amigable.

Configurable: debe ser posible a través de configuraciones, preferentemente en el estándar XML, definir los dispositivos habilitados para la captura, para que no sea necesario recompilar la aplicación en el caso que sea necesario usar otros dispositivos de otros fabricantes para realizar las mismas u otras operaciones de captura, y además se pueda crear una herramienta en la propia aplicación que sea capaz de gestionar este archivo de configuración de forma sencilla.

Análisis y Diseño

A partir de los requerimientos definidos para este *framework*, se realizó el análisis y el diseño de su estructura básica, teniendo en cuenta los aspectos más generales de cada tipo de dispositivo en particular, con el objetivo de cumplir las premisas de su funcionamiento.

Clasificación

Primeramente se analizaron los diferentes tipos de dispositivos presentados en la fase inicial de investigación, clasificándolos teniendo en cuenta sus características comunes y particulares en cada caso, para luego modelar un conjunto de clases que darían el soporte principal para el posterior desarrollo de los componentes específicos encargados de la interacción con los equipos y el software vinculado a estos. Las principales clasificaciones establecidas fueron:

Captador de firma: un dispositivo en forma de tabla o pizarra capaz de captar la firma de la persona y digitalizarla para luego ser procesada por una aplicación, y que puede o no mostrar la firma en la tabla de forma interactiva.

Captador de foto: dispositivo encargado de capturar la foto digital de la persona, y mostrar la imagen en tiempo real en la interfaz de la aplicación.

Captador de huella: el dispositivo que captura la huella de la persona de forma digital como imagen y que muestra el proceso de captura en tiempo real en la interfaz de la aplicación.

Lector de página completa: un equipo capaz de leer los datos de los documento de viaje especificados por la ICAO (*International Civil Aviation Organization*)[1], como Pasaporte o Visa y obtener los datos que se puedan leer de forma óptica, como el código OCR (*Optical character recognition*) [2] y la zona de inspección visual. De forma opcional puede leer código bidimensional, obtener imágenes del documento y leer los datos almacenados electrónica a través de un chip por contacto o por RFID (*Radio Frequency IDentification*) [3].

Programador de SmartCard (Tarjeta inteligente): el dispositivo encargado de leer y escribir datos en una tarjeta inteligente tanto por contacto como por RFID.

Modelación

Para modelar las clases base de este submódulo se definió un conjunto de interfaces para la fácil integración con los negocios en las aplicaciones del sistema. Estas interfaces definen el comportamiento general de todos los dispositivos en cuanto a funcionalidad, propiedades y eventos de notificación, luego se definieron un conjunto más grande de interfaces que implementan las anteriores y además agregan las características específicas de cada tipo de dispositivo según sea el caso.

Este *framework* fue desarrollado sobre Microsoft .Net Framework v1.1.4, en el lenguaje C# y con el entorno de desarrollo Microsoft Visual Studio .Net 2003.

En la siguiente figura se muestra el diagrama de interfaces y clases base para el diseño de este *framework*:

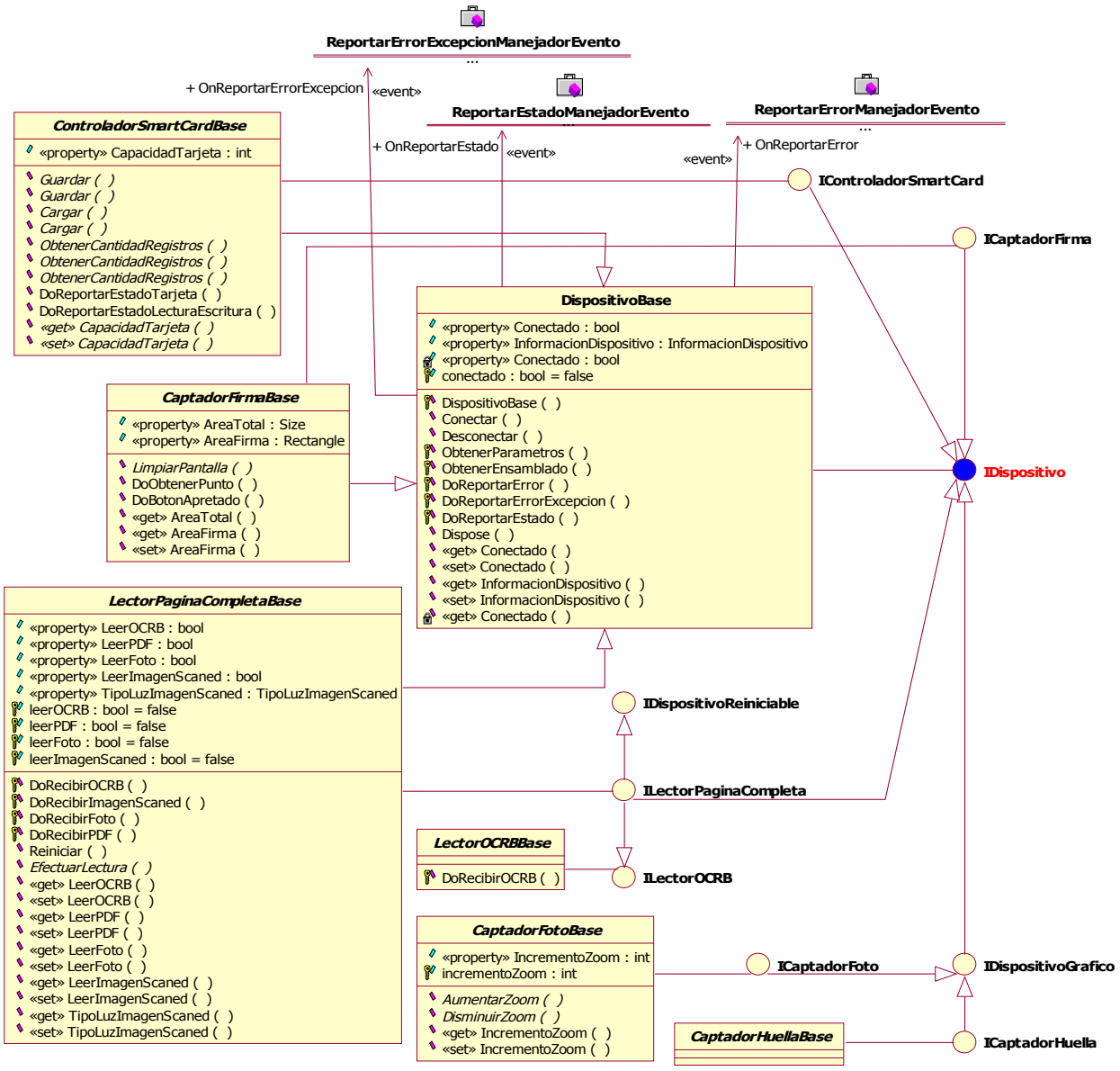

Fig. 1. Diagrama de clases del diseño.

En este diagrama se muestra la interfaz "**IDispositivo**", que describe el comportamiento general de cualquier tipo de dispositivo, como las operaciones de "**Conectar**" y "**Desconectar**", las propiedades de "**Conectado**" para saber si está conectado el dispositivo e "**InformacionDispositivo**" que provee un conjunto de datos para mostrar alguna información técnica del equipo, y sus principales eventos de notificación, como "**OnErrorExcepcion**" para cuando hay una excepción en el funcionamiento, "**OnReportarEstado**" que notifica de un cambio de estado. Implementando esta interfaz base, existen una serie de interfaces que especifican la descripción en comportamiento, eventos e información para cada tipo de dispositivo.

Para cada clasificación de dispositivo se definieron un conjunto de interfaces, según se presenta en la figura anterior, las cuales implementan la interfaz "**IDispositivo**". A continuación se muestra la descripción de una de estas interfaces, a través de sus funcionalidades, propiedades y eventos:

- ICaptadorFirma: Interfaz para los tipos de dispositivos que capturan la firma de la persona usando una tabla de firma.

- LimpiarPantalla: Método para limpiar la pantalla de la tabla de firma.

- AreaFirma: Propiedad que define las dimensiones del área de firma en el dispositivo.

- AreaTotal: Propiedad que define las dimensiones del área total de contacto del dispositivo.

- OnObtenerPunto: Evento que notifica cuando se obtiene un punto del captador de firma.

- OnBotonApretado: Evento que notifica cuando es presionada un área de la tabla de firma que es definida como un botón.

Implementación

Para poder implementar la comunicación con un dispositivo específico, con el objetivo de poder integrarlo a cualquier flujo de la aplicación a través de las interfaces definidas para su tipo, es necesario el estudio y dominio de un conjunto de tecnologías que desde distintos niveles de programación soportarán la interacción final. Estas tecnologías poseen niveles de abstracción que facilitan o no el desarrollo de componentes de integración, las más usadas y ordenadas desde las más complejas a las más simples son:

Tecnologías para la implementación

APIs de Windows: un conjunto de funcionalidades que brinda el Sistema Operativo Microsoft Windows en todas sus versiones, que permiten la comunicación y acceso a todo tipo de recursos controlados por este, con lo cual es posible implementar componentes para interactuar, en este caso con los dispositivos que se conecten a la estación por las distintas interfaces físicas más conocidas como: LPT, COM, USB y *Firewire*, usando protocolos como el RS-232. Estas APIs (*Application Programming Interface*) [4], están bien documentadas en la MSDN (*Microsoft Developer Network*) [5], con la explicación de cada función y ejemplos de uso.

Objetos COM y ActiveX: COM (*Component Object Model*) [6], formó los primeros intentos de Microsoft por lograr una plataforma para la implementación de componentes que pudieran ser usados independientemente del lenguaje en que fueron desarrollados para de esa forma poder ser usados en distintas aplicaciones y entornos a través de interfaces o "contratos" bien definidos que describían su estado y comportamiento. Los componentes ActiveX son una implementación más avanzada de esta plataforma de componentes que se comenzaron a usar principalmente en las páginas y aplicaciones Web para brindar más potencialidad a este tipo de entorno.

TWAIN: es un estándar para la adquisición de imágenes a partir de un dispositivo de captura como un escáner o una cámara digital, es independiente del sistema operativo y existen implementaciones para varias plataformas y tecnologías de desarrollo, permitiendo la comunicación con estos equipos de captura, el tratamiento de las imágenes capturadas y las configuraciones requeridas para el proceso de captura. [7]. La documentación sobre estas especificaciones y sus implementaciones están en el sitio de la organización que lo diseñó, y disponible en: http://www.twain.org/.

WIA: WIA (*Windows Image Acquisition*) [8], es un modelo de *drivers* o controladores y una API para la comunicación con dispositivos de adquisición de imágenes como escáner, cámaras digitales y equipos de video digital, no es multiplataforma como TWAIN ya que es solo para sistemas operativos Microsoft Windows en sus versiones más modernas, pero es más fácil de usar e integrar con aplicaciones ya que existen un conjunto de componentes principalmente ActiveX para la comunicación con este tipo de dispositivos además de documentación y ejemplos disponibles en la MSDN.

WinTab: es un estándar para la interacción con dispositivos de captura de firma, como las firmas convencionales pero de forma digital, las APIs que se han implementado no son multiplataforma, pero esta especificación si es estándar y abierta. Su principal objetivo es la fácil comunicación con este tipo de dispositivos y su integración con aplicaciones que lo requieran. Las especificaciones para el desarrollo de componentes y controladores bajo este estándar se encuentran en: http://www.sonycsl.co.jp/projects/ar/restricted/wintabl.html.

Microsoft .Net Interop: este es un mecanismo por el cual la Tecnología .Net puede hacer llamadas y uso de tecnologías anteriores o nativas como las DLL (*Dynamic Linking Library*) [9] desarrolladas sobre tecnologías más antiguas como C++, de ahí su nombre de "Interop" o interoperabilidad entre estas plataformas no administradas y .Net. A través del espacio de nombre o "namespace", "**System.Runtime.InteropServices**", se brindan un conjunto de clases y estructuras para este tipo de desarrollo y con el uso del atributo [**DllImport**("[nombre del archivo dll]")], sobre un método estático que defina la firma o estructura del procedimiento que se encuentre implementado en esta DLL y con el modificador **extern** ya será posible usar las funciones definidas e implementadas que contengan estos componentes, dentro del código administrado de .Net, por ejemplo:

```
[DllImport("kernel32.dll")]
static extern bool Beep (uint dwFreq, uint dwDuration);
```

Existen varios ejemplos de cómo implementar el mecanismo de Interop en la Tecnología .Net, además de una documentación sobre las principales APIs de Windows, disponibles desde la dirección: http://pinvoke.net/.

SDK de los fabricantes: el SDK (*Software Development Kit*) [10], es un conjunto de herramientas, APIs, documentación y usualmente ejemplos, en este caso para la comunicación con determinado dispositivo, que provee el fabricante para una cómoda integración de su producto con las aplicaciones en general. Este es el mecanismo más fácil de todos los anteriormente mencionados ya que los incluye a casi todos, y muchas veces brinda desde un nivel mas alto las funcionalidades que provee el dispositivo, o de lo contrario a través de las tecnologías anteriores se podrán usar las APIs que brindan estos SDK, y con la documentación que obligatoriamente debe aparecer, se podrá implementar el acceso a estos dispositivos.

Pasos para la implementación de nuevos dispositivos

Teniendo en cuenta el diseño de interfaces y clases anteriormente explicado y con el uso de las tecnologías descritas, ya se cuenta con todas las herramientas necesarias para la implementación de nuevos dispositivos que se necesiten en el sistema, tanto los de nuevo tipo como de los ya existentes pero de otros fabricantes o de tipo genérico, y teniendo en cuenta los siguientes pasos será posible completar la implementación.

Clasificar el dispositivo del que se quiere desarrollar su componente de integración o acople en dependencia de sus funcionalidades.

Crear una clase que herede de la clase específica en dependencia del tipo de dispositivo que se va a usar o implementar las interfaces necesarias en el caso de que el dispositivo combine las funcionalidades de varios de los tipos definidos, como ocurre en el caso del "**ILectorPaginaCompleta**" que surgió de la combinación de un "**ILectorOCRB**", un "**IDispositivoReiniciable**".

Si el dispositivo es totalmente nuevo, definir la nueva interfaz que debe siempre implementar de la interfaz "**IDispositivo**" y luego implementar esta interfaz en el dispositivo específico o crear una clase base para ese tipo de dispositivo que implemente la recién creada y la clase que interactúa finalmente con el dispositivo que debe heredar de esta.

- La clase que contendrá la implementación de la interacción con el dispositivo debe estar en una DLL separada para que pueda ser fácilmente integrable a la aplicación a través del mecanismo de carga dinámica para el dispositivo que se use en ese momento.
- Usar las tecnologías disponibles y anteriormente explicadas para implementar las funcionalidades que describen a este tipo de dispositivo.
- Tener en cuenta en las secciones del código donde es muy posible errores de funcionamiento y usar los bloques "try-catch-finally" para notificar los posibles errores ocurridos con el evento definido para esto y documentar todos los posibles errores para saber en cada momento donde existe una falla y cómo poder solucionarla.
- Tener en cuenta que el dispositivo puede pasar de un estado a otro en los procesos de lectura escritura o captura de datos e imágenes, para lo cuál se debe notificar estos cambios a través del evento definido para esto, y en caso de que el estado que se quiera especificar no aparezca entre los disponibles agregarlo al enumerador que corresponda.
- Probar todos los posibles estados por los que debe pasar el dispositivo en el proceso de captura para eliminar errores de implementación y soportar malas operaciones del usuario.
- Si el dispositivo requiere de un componente gráfico para la interacción con la interfaz de la aplicación, se debe implementar el control luego de haber hecho y probado la interacción con el dispositivo debido a que el control tiene una referencia al tipo de dispositivo que usa por lo que depende de este para su funcionamiento.

Ejemplo de implementación de un dispositivo de captura de firma

En este ejemplo aparecen implementados los pasos ya mencionados para el desarrollo e integración de un dispositivo de captura de firmas.

Luego de identificar que este dispositivo pertenece al tipo Captador de Firma, usando cualquier herramienta de desarrollo se crea un nuevo proyecto que contenga una clase, por ejemplo: CaptadorFirma, que herede de CaptadorFirmaBase o implemente la interfaz ICaptadorFirma previamente descrita, de esta forma ya tendremos todas las funcionalidades que debe brindar nuestro componente de acceso a dispositivo hacia la aplicación que estemos desarrollando. Una vez hecho esto la clase queda de la siguiente forma:

```
public class CaptadorFirma : CaptadorFirmaBase
{
	public CaptadorFirma()
	{
		//Constructor de la clase.
	}

	#region ICaptadorFirma Members

	public override bool LimpiarPantalla()
	{
		//Se implementa como se limpia la pantalla del dispositivo
		return false;
	}

	#endregion

	#region IDispositivo Members

	public override bool Desconectar()
	{
		if(Conectado)
		{
			//Se implementa cómo desconectar el dispositivo
			return  true;
		}
		return false;
	}
	public override bool Conectado
	{
		get
		{
			return conectado;
		}
	}

	public override bool Conectar()
	{
		if(!Conectado)
		{
			//Se implementa cómo conectar el dispositivo
			return true;
		}

		return false;
	}
```

```
        protected override ColeccionParametros ObtenerParametros()
        {
              //Se devuelven un conjunto de parámetros que contienen
              //información sobre el dispositivo
        }

        #endregion

        #region IDisposable Members

        public override void Dispose()
        {
              Desconectar();
        }

        #endregion
  }
```

Se debe tener en cuenta que en el caso de los dispositivos de captura de firma existen dos eventos que se deben notificar cada vez que el componente con el que se comunica, ya sea el SDK del proveedor o componentes estándares como **WinTab** en este caso, con el objetivo de que el control gráfico que muestra el proceso de captura de firma, pueda dibujar los puntos que se van obteniendo o pueda reaccionar ante la acción que realice la persona sobre uno de los botones definidos en el área de contacto de la tabla de firma a través de los eventos: "**OnObtenerPunto**" y **"OnBotonApretado", a**demás de notificar el evento "**OnErrorExcepcion**" ante la ocurrencia de un error en el proceso de captura y el cambio de estado del dispositivo al conectarlo o desconectarlo con el evento "**OnReportarEstado**".

Una vez implementada esta clase, teniendo en cuenta todos los aspectos anteriores, solo es necesario compilar el ensamblado y generar el archivo DLL para poder usarlo en la aplicación a través del mecanismo de configuración y carga dinámica.

Despliegue

Descripción del mecanismo de configuración

Para lograr que los dispositivos que sean usados por la aplicación puedan ser cambiados y reemplazados sin tener que recompilar la aplicación para una nueva versión de esta, además del uso de interfaces generales que describen el comportamiento de cada tipo de dispositivo específico, también se implementó un mecanismo general para integrarlos a la aplicación final de una forma sencilla y fácil de configurar.

A través de la creación de un archivo de texto en el estándar XML, se puede describir los tipos de dispositivos que se usan en la aplicación actual y cuáles de estos se están usando en el momento. A continuación se muestra un ejemplo de configuración para una aplicación que usa un dispositivo de captura de firma, y luego se describe cada uno de los "tags" o etiquetas que lo componen:

```
<ConfiguracionDispositivo>
 <Dispositivos>
```

```
    <Dispositivo>
      <Tipo>CaptadorFirma</Tipo>
      <Controladores>
        <Controlador Activo="true">
          <Descripcion>CaptadorFirma general que detecta el que esta conectado</Descripcion>
          <Ensamblado>Dispositivos.CaptadoresFirma.EpadInk.dll</Ensamblado>
          <NombreClase>Dispositivos.CaptadoresFirma.EpadInk.CaptadorFirma</NombreClase>
        </Controlador>
      </Controladores>
    </Dispositivo>
  </Dispositivos>
</ConfiguracionDispositivo>
```

- **ConfiguracionDispositivo**: es la etiqueta que contiene la configuración de todos los dispositivos que son usados por la aplicación.
- **Dispositivos**: contiene la lista de etiquetas "Dispositivo".
- **Dispositivo**: contiene la información de todos los dispositivos de un tipo que se conecten a la estación e interactúen con la aplicación.
- **Tipo**: define el tipo de dispositivo.
- **Controladores**: contiene la lista de etiquetas "Controlador".
- **Controlador**: define los datos para un controlador de un tipo de dispositivo conectado a la estación. Tiene la propiedad "**Activo**", que define si este controlador va a estar activo en ese momento en la aplicación y puede tomar valores de "**true**" o "**false**", es decir que sea ese o no el controlador que se va a usar para que la aplicación se comunique con ese tipo de dispositivo, debido a que si hay varios dispositivos de un mismo tipo conectados a la estación, solo puede haber un controlador activo, es decir solo puede haber un dispositivo de un tipo funcionando en la aplicación.
- **Descripción**: contiene una breve descripción del controlador actual.
- **Ensamblado**: esta etiqueta es muy importante porque define el nombre exacto del archivo DLL que contiene la clase que interactúa con el dispositivo, ejemplo: "Dispositivos.CaptadoresFirma.EpadInk.dll".
- **NombreClase**: contiene el nombre completo de la clase que implementa la interacción con el dispositivo, ejemplo:

"Dispositivos.CaptadoresFirma. EpadInk.CaptadorFirma"

Al ser muy fácil modificar este archivo para agregar, cambiar o eliminar dispositivos que se estén usando actualmente, debido a la fácil comprensión de un formato simple de texto y muy descriptible de su contenido como es el estándar XML y a la cantidad de componentes y tecnologías existentes que permiten el acceso a este tipo de formato, la aplicación que lo usa pudiera proveer una interfaz amigable para la gestión sobre este archivo de configuración, aunque siempre puede ser modificado en cualquier editor de texto simple.

Integración de nuevos dispositivos

Incluida en este *framework* de acceso a dispositivos, existe una clase llamada "**ControladorDispositivo**", que es la encargada de cargar "**dinámicamente**" el componente DLL que interactúa con el dispositivo actual según se especifique en el archivo de configuración, esto es posible a través del **namespace** "**System.Reflection**" del

framework de .Net, que contiene las clases necesarias para la carga dinámica de DLLs para instanciar la clase del dispositivo y de esa formas lograr la fácil integración de nuevos dispositivos sin tener que recompilar la aplicación que los utiliza. Para lograr que esta ejecución dinámica funcione bien y que el dispositivo interactúe de igual forma con la aplicación es necesario tener en cuenta algunos requisitos:

- El "*driver*" o controlador del dispositivo actual debe estar instalado y se debe comprobar a través de las aplicaciones de ejemplo que usualmente vienen con esta instalación, que el equipo funcione bien.
- Se debe copiar el archivo DLL que contiene la clase que interactúa con el dispositivo, en la raíz de la aplicación, para de esta forma poder ser cargada dinámicamente por el *framework* de acceso a dispositivos.

Especificar correctamente los datos que se requieren en el archivo de configuración de los dispositivos para que no ocurran errores en el momento de instanciar la clase que se comunica con el equipo, debido a que a partir de esta configuración se va a realizar la carga de los dispositivos que se usan en ese momento.

Si ocurre algún error en la aplicación cuando se llega a la parte en que se interactúa con el equipo para hacer alguna lectura, escritura o captura de datos, se debe comprobar los tres aspectos anteriores, de seguir el problema, comunicarse con el equipo de soporte, para un mejor estudio del problema.

Observaciones

Este *framework* a pesar de haber sido desarrollado sobre la plataforma .Net de Microsoft, podría ser implementado también sobre tecnologías .Net *Open Source* como el Proyecto Mono y DotGNU que han sido elaborados para distintos Sistemas Operativos, o también se podría desarrollar todo el *framework* en tecnologías multiplataforma como Java o Python.

Una desventaja de la idea de este *framework* para integrar el acceso a dispositivos, es que los "*drivers*" o controladores y los SDK de los equipos para determinados fabricantes, solo brindan las APIs para la interacción sobre la plataforma Windows y no sobre otros Sistemas Operativos, fundamentalmente libres, por lo que si implementamos esta solución para otra plataforma, necesitamos crear nuestros propios componentes de interacción con dispositivos a más bajo nivel comparado con el que podría brindarnos un SDK.

Conclusiones

Con el desarrollo de este trabajo se ha profundizado en el análisis de las características comunes en los distintos dispositivos que existen en el mercado, para determinados procesos en los que cualquier sistema o aplicación podría usarlos para la captura de datos, imágenes o comprobación de documentos.

Se ha podido proveer un mecanismo fácil y transparente para el desarrollo de módulos o aplicaciones que necesiten la interacción con estos dispositivos, y la posibilidad de extender funcionalidades o integrar nuevos equipos con

características similares que cumplan los requisitos de calidad, para una mejor estrategia de despliegue en función de la disponibilidad existente.

Este *framework* que se ha diseñado e implementado, es independiente de cualquier plataforma, debido a que solo propone un esquema y metodología para la integración de aplicaciones con dispositivos externos, lo cual asegura la posibilidad de ser desarrollado sobre otras tecnologías con características similares a la actualmente usada.

Referencias Bibliográficas

[1] Wikipedia, Organización de Aviación Civil Internacional, 2008. [Disponible en: http://es.wikipedia.org/wiki/ICAO]

[2] Wikipedia. Reconocimiento óptico de caracteres, 2008. [Disponible en: http://es.wikipedia.org/wiki/OCR]

[3] Wikipedia. RFID, 2008. [Disponible en: http://es.wikipedia.org/wiki/RFID]

[4]Wikipedia. Application Programming Interface, 2008. [Disponible en: http://es.wikipedia.org/wiki/Application_Programming_Interface]

[5] Wikipedia. Microsoft Developer Network, 2008. [Disponible en: http://en.wikipedia.org/wiki/MSDN]

[6] Wikipedia. Component Object Model, 2008. [Disponible en: http://es.wikipedia.org/wiki/Component_Object_Model]

[7] Wikipedia. TWAIN, 2008. [Disponible en: http://es.wikipedia.org/wiki/TWAIN]

[8] Wikipedia. Windows Image Acquisition, 2008. [Disponible en: http://es.wikipedia.org/wiki/Windows_Image_Acquisition]

[9] Wikipedia. Dynamic Linking Library, 2008. [Disponible en: http://es.wikipedia.org/wiki/DLL]

[10] Wikipedia. SDK, 2008, [Disponible en: http://es.wikipedia.org/wiki/SDK]