Universitat Autònoma de Barcelona

2432: Kiosk Digital Online

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Iban Benzal Muñoz
i dirigit per
Enric Martí Godia
Bellaterra 9 de Setembre de 2010

Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

El sotasignat, Enric Martí Gòdia

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat:

Bellaterra, 9 de setembre de 2010

CONTENIDO

1. INTRODUCCIÓN

El uso de las aplicaciones ricas de Internet (*Rich Internet Applications*, a partir de ahora RIA) nacieron para mejorar la experiencia del usuario en Internet(Adobe, 2010). Las páginas webs tradicionales limitan la interactividad del usuario con la web, con estructuras estáticas y poca respuesta de los objetos, además de la pobre presentación a nivel gráfica y de animaciones que ofrecen.

Se creó una combinación de las ventajas que ofrecen las aplicaciones Web y las de escritorio, como podemos ver en la figura 1. Estas nuevas aplicaciones, llamadas RIA, poseen muchas de las características de las aplicaciones de escritorio, requieren de un "navegador web" para ejecutarse y un "plugin" o máquina virtual para agregar las características adicionales.

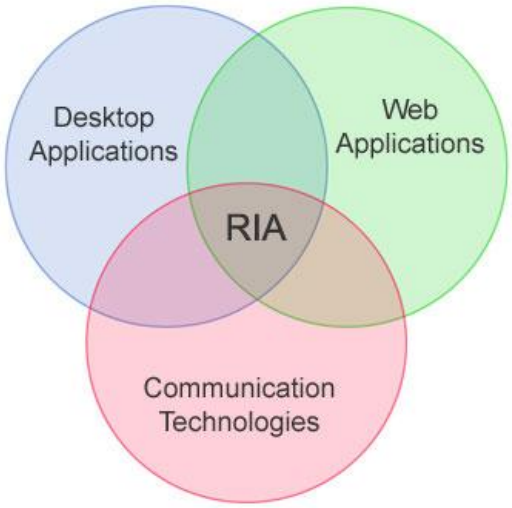

Figura 1: Componentes de RIA.
Extraída de (Oikoumene, 2010)

Frente las aplicaciones web tradicionales, podemos nombrar una serie de ventajas de las RIA:

- Desaparición de la recarga de la página cada vez que el usuario pulsa sobre un enlace. En los entornos RIA no se producen recargas de página, ya que desde el principio se carga toda la aplicación, y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una Base de Datos o de otros ficheros externos.
- Uso de contenidos multimedia.
- No se precisa de instalación.
- Uso desde cualquier ordenador con una conexión a Internet sin depender del sistema operativo que este utilice.
- Aplicaciones interactivas que no se pueden obtener utilizando sólo HTML, incluyendo arrastrar y pegar, procesado en el lado del cliente sin la necesidad de enviar la información al servidor.

- Evita la problemática del uso de diferentes navegadores al abstraerse de ellos a través de un framework.
- Para que las RIA puedan comunicarse con el servidor, requieren tecnologías avanzadas de comunicación, por ejemplo: protocolos de internet optimizados, conexiones asíncronas, pre-fetching data (como en Google Maps(GoogleMaps, 2010)). Por lo tanto, las muchas RIA requieren de conexiones de gran ancho de banda.

Mitsubishi Electric Photo (Mitsubishi, 2010) se interesa por estas tecnologías y encarga un estudio sobre el desarrollo de las aplicaciones RIA. Antes de dicho estudio se seleccionará una herramienta de desarrollo de RIA entre las diferentes del mercado.

Una vez seleccionada la herramienta, empezará el diseño de un conjunto de componentes orientados a facilitar el desarrollar aplicaciones en esta RIA. Estos componentes serán un Framework de desarrollo para las futuras aplicaciones.

El Framework es muy importante ya que dotaran a aplicaciones de modularidad, escalabilidad y del desglose de la interfaz de usuario con el modelo de datos. Además que usar este Framework implicará un considerable ahorro de tiempo a diferencia de empezar la aplicación desde cero.

Siguiendo con las RIA, hoy en día existen muchas herramientas para la creación de entornos RIA, entre éstas se puede mencionar las plataformas Flash, Flex y AIR de Adobe (Adobe, 2010), Silverlight (Silverlight, 2010) de Microsoft y JavaFX(JavaFX, 2010) de Sun Microsystems.

En la siguiente figura tenemos una tabla para introducirnos en cada RIA:

Nombre	Empresa	Publicación	Versión actual	Lenguaje	Integración
Flex	Adobe	2004	4	ActionScript 3.0	Flash Player
Silverlight	Microsoft	2007	4	Lenguajes de .NET	Silverlight Plugin
JavaFX	Sun	2008	1.3	Java	Máquina Virtual

No existe una RIA superior al resto puesto que cada una tiene ventajas y desventajas, y dependiendo el proyecto a realizar quizá interesa premiar más al apartado visual, que a la orientación de objetos, por ejemplo.

Adobe Flex es un RIA muy maduro y que posee un apartado gráfico, sobretodo en animaciones, increíble. En cambio su lenguaje, ActionScript3, tiene una serie de limitaciones que le restan puntos para orientación a objetos; no se pueden declarar constructores privados (necesario para Singleton).

JavaFX flojea por su falta de madurez y su framework, orientado a teléfonos móviles, provoca que las aplicaciones Web tengan ciertas deficiencias.

Silverlight se ha convertido versión tras versión en un RIA más serio. Al estar dentro del framework .NET puede ser programado en C#, VB.NET, IronPython o IronRuby, además

de tener acceso a un gran conjunto de librerías profesionales. Su apartado gráfico tiene mucha versatilidad al usar DirectX.

En cuanto su implementación en clientes finales:

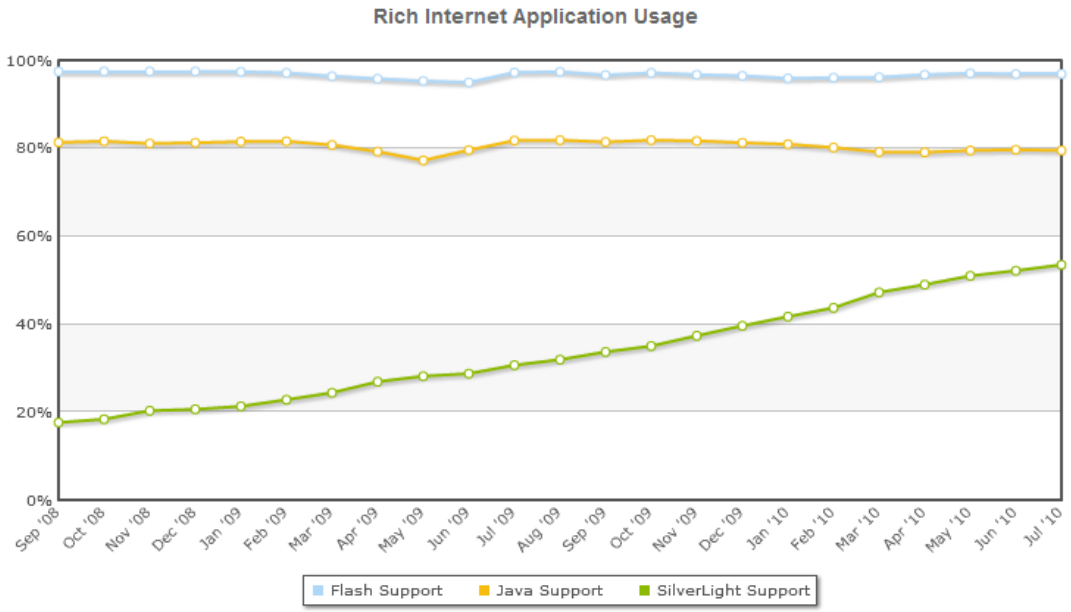

Figura 2: Ordenadores que soportan RIA.
Extraída de (Statowl, 2010)

En la figura 2 podemos ver que casi todas las conexiones a Internet soportan Flash, un 80% soportan el applet de Java y que Silverlight está presente en el 53% de las conexiones. Esto demuestra que las RIA han conseguido una gran importancia en Internet, y que es un sector de constante crecimiento. Desde la página web de la empresa hasta tiendas virtuales, la versatilidad de las RIA permite llegar a un grupo de clientes y desarrolladores muy amplio.

En este proyecto se va a usar Silverlight. La balanza se ha inclinado hacia este lado debido a:

- El Framework .NET (NET, 2010), que ya está en su 4 versión. Silverlight se va en este Framework como podemos ver en la Figura 3. De esta manera, además de poder utilizar librerías de Microsoft, se puede reutilizar código en otros proyectos.
- C#, VB.NET, *IronPython* o *IronRuby* para el acceso a los objetos.
- XAML[1] permite una gran versatilidad a la hora de crear componentes para la interfaz de usuario y posee acceso a un motor de *DirectX* para crear animación.
- Integra en un solo complemento: multimedia, animaciones e interactividad.
- Como hemos visto en la figura 2, esta RIA cada vez está más expandida entre los usuarios.

[1] XAML es un lenguaje declarativo basado en XML, optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico (Wikipedia, 2010)

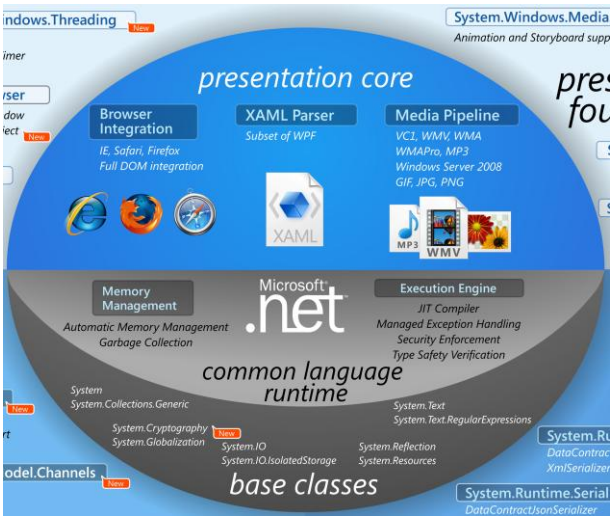

Figura 3: Núcleo de Silverlight
extraída de (Silverlight, 2010)

El objetivo del proyecto es realizar un estudio de Silverlight diseñando e implementado un Framework que permita crear aplicaciones con las siguientes características:

- Respetar el modelo de datos, que no se mezcle con la interfaz del usuario.
- Basada en plugins: los módulos de la aplicación deben ser cargadas bajo demanda.
- Escalabilidad y modulación.
- Comunicación entre módulos: permitir mandar mensajes u objetos entre clases.
- Uso de .NET 4 (NET, 2010).

Este Framework se utilizará para realizar un prototipo donde se pueda demostrar la capacidad de éste y la interactividad del usuario con Silverlight, tiendo en cuenta que la aplicación final irá destinada al relevado digital.

Las motivaciones principales del proyecto son:

- El interés de la capacidad real de Silverlight, además que la creación de un Framework de desarrollo en Silverlight puede dar lugar a una gran cantidad de aplicaciones comerciales.
- Personalmente, la posibilidad de aprender un entorno de desarrollo de RIA en crecimiento, como lo es Silverlight.

A continuación se nombrarán las herramientas de software que han hecho posible el desarrollo del proyecto:

- Visual Studio 2010 (VS2010, 2010)
- Expression Studio (ExpressionStudio, 2010)

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows. Al ser el único entorno de desarrollo capaz de diseñar e implementar aplicaciones del framework .NET 4.0, no tenía competidores.

Soporta de manera nativa el desarrollo de aplicaciones en Visual C# y el código XAML de Silverlight.

Gracias al acuerdo de Microsoft con la UAB, los estudiantes tenemos el acceso gratuito a la versión ULTIMATE de este software.

Por otro lado, *ExpressionStudio* es una suite de herramientas de diseño y contenido multimedia desarrollado por Microsoft. En nuestro caso, de esta suite usamos Expression Blend, una herramienta para ayudar al desarrollador de XAML.

Entre sus múltiples ventajas están las de ver cómo quedará la vista sin necesidad de ejecutar el código, autogenerar los códigos más comunes y la compatibilidad con Visual Studio 2010, permitiendo modificar el mismo fichero con ambos programas a la vez.

A continuación se realizará una breve explicación del contenido de cada uno de los puntos de los que está formado el proyecto.

El capítulo 2, **Desarrollo**, contiene la información técnica del desarrollo del de cada uno de los objetivos nombrados. Explicará todo lo relacionado con los módulos Framework, desde por qué se ha diseñado así hasta su arquitectura. Se usarán diagramas en UML y se evitará el código fuente.

El capítulo 3, **Resultados**, se explicará el prototipo, cómo y dónde han sido utilizados los diferentes módulos del Framework, para verificar su funcionamiento, e imágenes y explicación del comportamiento.

El capítulo 4, **Conclusiones y Mejoras**, en este apartado se hará una reflexión sobre el proyecto, dando como resultado un listado de incidencias, posibles mejores y futuras funcionalidades.

El capítulo 5, **Bibliografía y Referencias**, contendrá las referencias bibliográficas de las cuales se han extraído directamente o indirectamente información para el desarrollo del proyecto.

2. DESARROLLO

2.1 FRAMEWORK

A la hora de desarrollar el framework tenemos que cumplir dos objetivos que una aplicación con un buen diseño no debe pasar por alto: escalabilidad y modulación. Además, al tratarse de una aplicación que pretende potenciar la experiencia de usuario necesitamos un desglose del modelo de datos con la vista.

En la siguiente figura se muestra el diagrama de módulos que tiene el framework desarrollado:

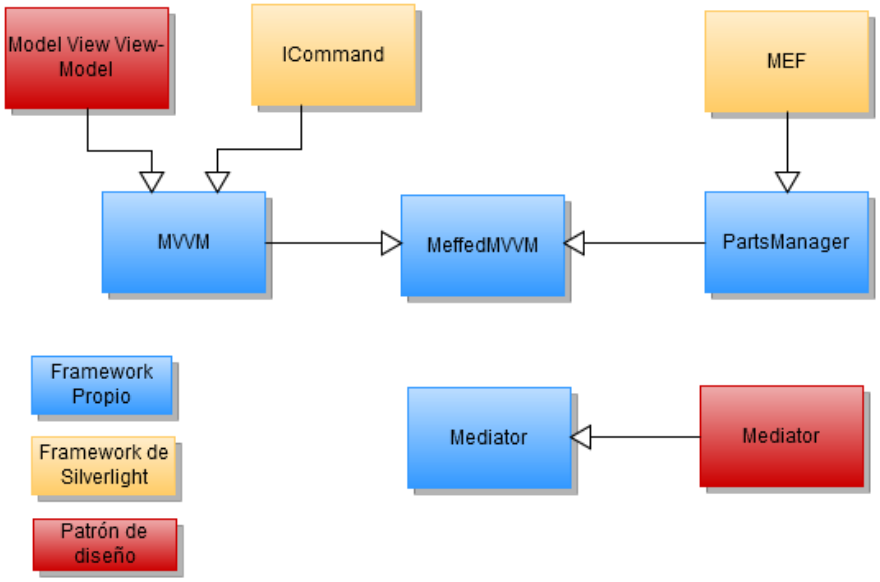

Figura 4: Framework

Se han desarrollado los siguients módulos:

- *MVVM*, que aplica el patrón de diseño *Model View ViewModel*, sirve para separar la interfaz de usuario del modelo de datos. Este módulo define cómo se deben declarar las ventanas de la aplicación. Requiere de *ICommand* (MSDN, 2010).
- *PartsManager* es el compontente que aporta a las aplicaciones la modularidad. El módulo implementa a la aplicación la capacidad de ser cargada mediante plugins y se delega en él el conocimiento del todas las clases del sistema, gracias a Managed Extensibility Framework (MEF, 2010) presente en el Framework de Silverlight.
- *MeffedMVVM* es el resultado de unir *PartsManager* y *MVVM*. Este módulo conoce todas las ventanas y es el único capaz de llamarlas.

- *Mediator* implementado a siguiendo el patrón de diseño con el mismo nombre (Erich Gamma, 1994) (Bishop, 2007), permite la comunicación entre módulos o objetos.

2.1.1 MODEL VIEW VIEWMODEL

Para conseguir la separación del modelo de datos con la vista, existe un patrón recomendado por Microsoft llamado *Model View ViewModel* (a partir de ahora *MVVM*). Por lo tanto, este módulo define cómo se deben declarar todas las ventanas de la aplicación.

MVVM (Smith, 2010) (Wikipedia, 2010) está basado en el patrón *Model View Controller* (MVC) (Wikipedia, 2010) y su función es ayudar al desarrollador que busca una experiencia de usuario más exigente que los desarrolladores tradicionales (*Back end*[2] o *Business Logic*). Como es en el caso de las plataformas de desarrollo de UI modernas como Silverlight o WPF.

Usar *MVVM* implica usarlo tanto en la capa de modelo de datos como en la interfaz de usuario. Es decir, nos olvidamos de la creación tradicional de ventanas que mediante eventos iban accediendo al modelo de datos. *MVVM* usa el *View*, que sería el equivalente a la funcionalidad visual de la ventana, *Model* que es el modelo de datos y el *ViewModel* que interactúa con los otros dos.

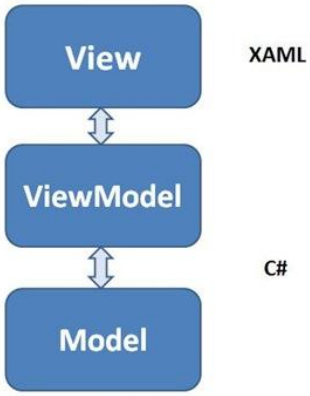

Figura 5: MVVM en una imagen

Si se tuviera que definir en una frase diríamos que *ViewModel* se encarga de exponer en el *View* los objetos del *Model* con una gestión de objetos sencilla y efectiva, como podemos observar en la figura 5.

De manera más técnica, en *MVVM*:

[2] Hace referencia a la visualización del usuario navegante por un lado (Front-end), y del administrador del sitio con sus respectivos sistemas por el otro (Back-end).

- La *View* está asociada al contexto de datos del *ViewModel*, así la *View* puede enlazarse fácilmente a las propiedades del *ViewModel*.
- El *ViewModel* refleja los cambios al *Model* llamando a métodos del *Model*.
- El *ViewModel* expone *ICommand* (MSDN, 2010) a la *View* para poder ejecutar acciones.
- El *Model* notifica de los cambios al *ViewModel* lanzando eventos.

El patrón *MVVM* permite tener una gran flexibilidad para poder cambiar la interfaz de usuario sin tener que reorganizar la lógica. Esto significa que usando *MVVM* los aspectos de diseño de interfaz y desarrollo de modelo de datos se pueden desarrollar de manera simultánea y por diferentes personas. Además, ayuda a tener componentes reutilizables para dentro y fuera del proyecto.

2.1.2 PARTSMANAGER

Seguimos de cerca el desarrollo de *Managed Extensibility Framework* (MEF) desde su *Preview* 6 (Abrams, 2009). A partir de esa versión el desarrollo de aplicaciones modulables había llegado a Silverlight.

Managed Extensibility Framework (MEF, 2010) es una librería que simplifica el diseño de aplicaciones extensibles. Esta nueva librería de Microsoft, que viene incluida por primera vez en el Framework .NET 4, ofrece la capacidad de descubrimiento y composición de ensamblados. De esta manera, ganamos en la modulación y escalabilidad del sistema y ayudamos a las third-parties y programadores externos interesados en desarrollar sobre nuestro aplicativo.

Pero MEF no es el primer Framework que se ideó, PRISM(PRISM, 2010) es un sistema de construcción modular de aplicaciones compatible con la versión de .NET 3.0 y posteriores. La principal desventaja que tenía PRISM es que inicialmente no recibió ningún tipo de apoyo de Microsoft y el desarrollo fue torpe y lento.

Hoy en día, cómo Glenn Block[3] explica en un artículo de su blog (Block, 2010), pese que MEF es suficiente para desarrollar aplicaciones extensibles, MEF y PRISM pueden convivir y son complementarios.

MEF trabaja como un modelo plugin: descargando los ensamblados, cargándolos en tiempo de ejecución y descubriendo las clases. Para tener una idea del patrón en forma de imagen podemos ver la figura 6.

Para poder gestionar las diferentes partes, en el Framework se ha implementado un módulo llamado *PartsManager*. Este módulo realiza lo básico; llamadas a MEF para componer partes, además de una serie de nuevas de funcionalidades:

- Inicializar un único catálogo de composición, donde cualquier elemento del aplicativo podrá buscar partes.

[3] Encargado del proyecto MEF.

- Descargar partes sin importar el origen, gracias a la abstracción, *PartsManager* puede componer desde la máquina local, desde el servidor remoto de la aplicación Siverlight o desde cualquier servidor en Internet.

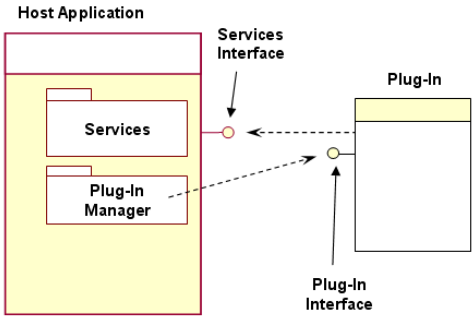

Figura 6: Modelo de plugin

- Componer y descomponer partes. Además de componer se ha añadido la funcionalidad de descomponer partes para ahorrar memoria y hacer la ejecución de la aplicación más liviana.
- Implementa un patrón *Singleton* (Bishop, 2007)(Erich Gamma, 1994) debido a la necesidad de tener una única instancia y proporcionar un punto de acceso global a ella. Así, incluso las third-Parties pueden componer.

2.1.3 MEFFEDMVVM (MEF + MVVM)

Llegados a este punto toca decidir cuánto vamos a involucrar MEF en el sistema. En un principio parece que sólo funciona como un modelo plugin, pero la capacidad de recomposición automática[4] abre muchas posibilidades.

Por ejemplo, diseñar un componente capaz de tener un listado con todas las ventanas disponibles de la aplicación. Así podemos delegar la lógica de obtener las ventanas a un componente. Como usamos *MVVM*, para obtener la ventana lo único que necesitamos el *ViewModel*.

La funcionalidad principal que se busca en este módulo es crear un componente *Indexer* capaz de indexar todas las ventanas del aplicativo mediante MEF. Así el conocimiento de las ventanas (*ViewModels*) como su creación se delega en este componente. Para ello el componente permite:

- Obtener cualquier ventana del aplicativo.
- Obtener una copia nueva del ventana, usando un el patrón *Factory,* figura 7, (Erich Gamma, 1994)(Bishop, 2007) o un copia previamente abierta.

[4] AllowRecomposition: Permite ir recomponiendo la clase si ha aparecido un objeto nuevo que cumple el contrato.

- Recomponer el listado de ventana según se vayan añadiendo nuevas partes. Esto se realiza gracias a *AllowRecomposition* (MEF, 2010).
- Crear la ventana justo cuando se vaya a mostrar, no cuando se añada a la lista. Clase *Lazy* (MSDN, 2010).

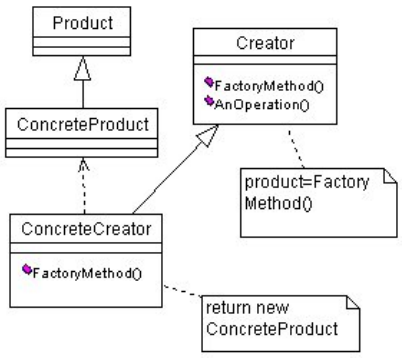

Figura 7: Patrón de diseño de Factory

2.1.4 MEDIATOR

En muchas ocasiones es necesario que haya una comunicación entre los componentes de la aplicación; transferencia de objetos, activación o desactivación de funcionalidades, refresco de variables… Para cubrir esta necesidad se ha diseñado un *Mediator*, implementando el patrón Mediator (Bishop, 2007)(Erich Gamma, 1994) (figura 8).

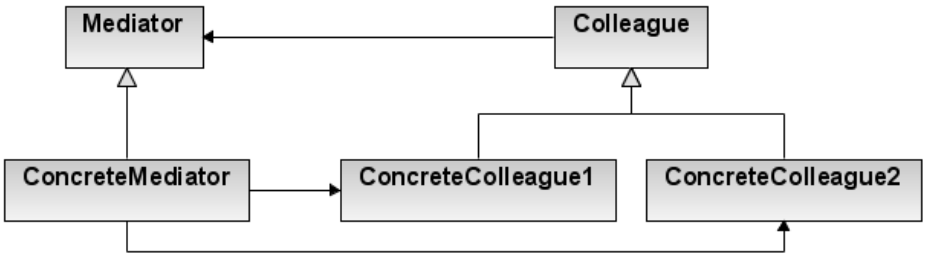

Figura 8: Patrón de diseño Mediator

El objetivo del Mediator es encapsular cómo interactúan un grupo de objetos entre ellos. Su funcionalidad es la siguiente, los componentes van asociando acciones a un mensaje y cuando un componente manda ese mensaje las acciones se ejecutan.

El diseño del Mediator requiere un modelo para representar la **acción**, un **componente que relacione las acciones con los mensajes** y el propio **Mediator**, que es el componente que encapsula las otras dos.

Para representar la **acción** se ha diseñado un componente que almacena que objeto ejecuta la acción y la acción. Este componente, llamado *WeakAction*, utiliza *WeakReference*(Mayo, 2001) (MSDN, 2010) para almacenar la dirección del objeto que ejecuta la acción. Se ha diseñado así para evitar que se quede memoria referenciada por objetos que no se necesitan. Es decir, que si el *Garbage Collector* (Mayo, 2001)de la aplicación va a recolectar un objeto y éste está referenciado en una acción del Mediator, lo pueda recolectar sin problemas. De esta manera evitamos *Memory Leaks* (Mayo, 2001) (Wikipedia, 2010) .

El componente encargado de **asociar acciones a mensajes**. Utiliza una tabla *Hash* (Mayo, 2001)(Wikipedia, 2010) para este propósito y además de añadir acciones, tiene la lógica de obtener las acciones y encapsularlas a delegados[5].

La lógica para añadir y obtener acciones está representada en los diagramas de estados representados en las figuras 9 y 10.

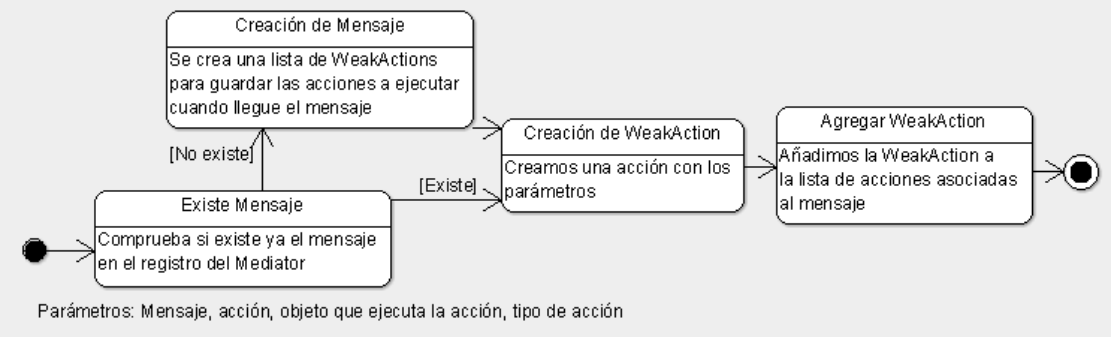

Figura 9: Diagrama de estados para añadir una acción

Como se puede apreciar en la figura 9, cuando se añade una acción al *Meditor*, primero se comprueba si el mensaje al que se quiere asociar la acción existe. En el caso de que ya exista, la acción se debe incluir en la lista donde están el resto de acciones para ese mensaje, en caso contrario se creará una nueva lista de acciones para el mensaje y ésta será su primera acción. Como trabajamos con *WeakActions*, debemos encapsular la acción en una WeakAction antes de añadirla a la lista.

En el caso que se quiera obtener un conjunto de acciones a partir de un mensaje, como se puede observar en la figura 10 primero se comprueba que exista el mensaje en el Mediator. Si existe, se obtienen las acciones y se crean los delegados. Antes de ejecutar los delegados se debe comprobar que el objeto que invoca de la acción exista, en caso contrario se debe eliminar la acción ya que no se podrá ejecutar nunca. Finalmente, si existe el invocador, ejecutamos la acción.

Gracias al componente que acabamos de mencionar, Mediator simplemente se limita a utilizarlo para añadir y obtener las acciones. Una vez obtiene las acciones las ejecuta una a una.

[5] *Delegate*: Tipo de clase de Silverlight que permite ejecutar acciones (MSDN, 2010).

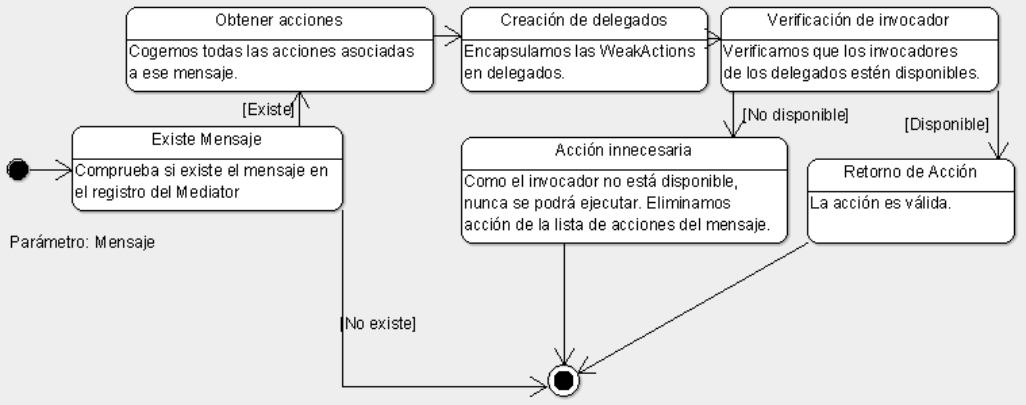

Figura 10: Diagrama de estados para obtener acciones

Mediator está encapsulado en un *Singleton* (Bishop, 2007)(Erich Gamma, 1994) para que pueda ser llamado desde cualquier parte del código y sólo exista una única instancia.

2.2 CONTROLES

Para poder verificar la capacidad de Silverlight en cuanto la experiencia de usuario a nivel de aplicativos de fotografía, se ha creado un conjunto de componentes definidos como Controles de Usuario, como se puede ver en la figura 11. De momento se han diseñado 2 controles:

- *AdornerArea*
- *DrawableArea*

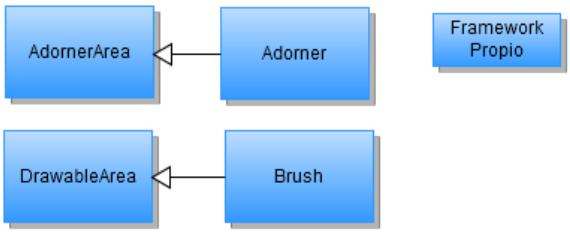

Figura 11: Controles

2.2.1 DRAWABLEAREA

DrawableArea es un control, que como su nombre indica, permite que se pueda dibujar sobre él. Se puede crear en cualquier ventana o control de usuario de

Silverlight o WPF, y no necesita de code-behind, sino que basta con definirse en el XAML.

Cuando *DrawableArea* captura el ratón tras un clic, le pasa a su objeto *Brush* las coordenadas por donde se está desplazando para que vaya dibujando el trazo. Obviamente el objeto *Brush* puede ir cambiando a lo largo de la ejecución.

El trazo se va dibujando sobre un objeto *WritableBitmap* (MSDN, 2010) del *DrawableArea*.

IBrush requiere de 4 métodos: para empezar a dibujar, para parar de dibujar, para dibujar y el que especifica que objeto será el *DrawableArea*.

Para facilitar la faena a los desarrolladores se ha implementado una clase base, como aparece en la figura 12, que implementa los 4 métodos de *IBrush* y permite sobrescribirlos (Virtual Methods en (MSDN, 2010)).

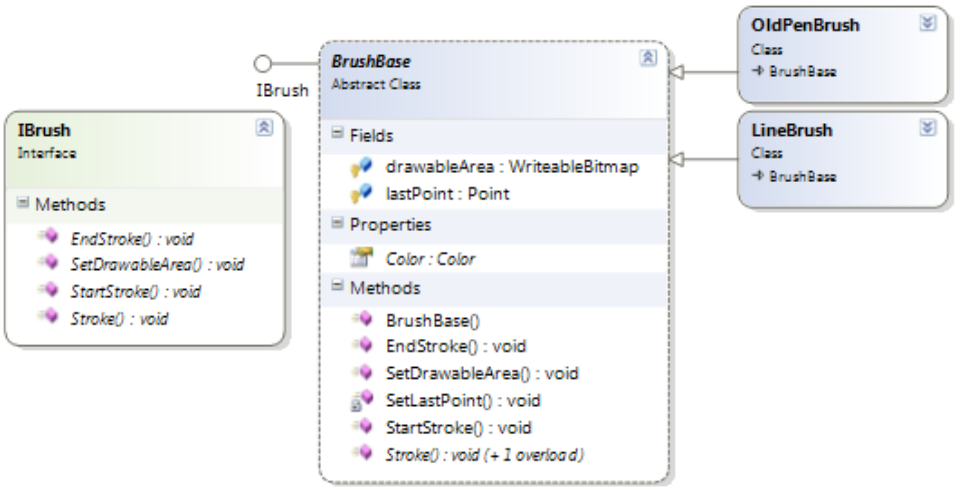

Figura 12: Diagrama de clases del modelo de *Brush*

Se ha añadido la propiedad Color, un método abstracto *Stroke* que ofrece más parámetros y limitamos el *DrawableArea* a *WritableBitmap* (MSDN, 2010).

2.2.2 ADORNERAREA

Microsoft introdujo un concepto muy interesante en WPF llamado *AdornerLayer* (MSDN, 2010). Consiste en una capa que se dibuja por encima de la capa principal de la interface de usuario y que puede contener cualquier elemento visual.

Esta capa no existe en Silverlight ya que Microsoft no la ha incorporado y no se sabe si en un futuro se incorporará.

El segundo control que se ha realizado se llama *AdornerArea* y simula la funcionalidad del *AdornerLayer*. Se puede crear en cualquier ventana o control de usuario de

Silverlight o WPF, y no necesita de code-behind, sino que basta con definirse en el XAML.

AdornerArea permite añadir objetos, que estos sean arrastrados, rotados, redimensionados y eliminados. Cuando se selecciona un objeto dentro de la *AdornerArea* le aparece un *frame* alrededor para diferenciarlo del resto. Tras añadir un objeto quedaría como en la figura 13.

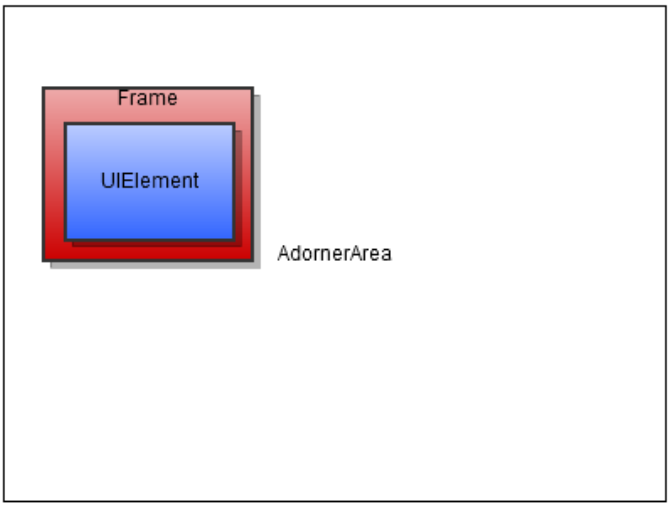

Figura 13: AdornerArea

El *frame* puede ser cambiado en tiempo de ejecución y los desarrolladores pueden crear nuevos *frames*.

Cuando un objeto, que debe heredar de *UIElement* (MSDN, 2010), es añadido a AdornerArea se crea un control *Adorner*. *Adorner* contiene el *UIElement* y es el que *gestiona* todas las acciones sobre éste. Por ejemplo, si el usuario estira el *Adorner*, éste le cambia el tamaño al *UIElement* dependiendo de la distancia que lo haya estirado.

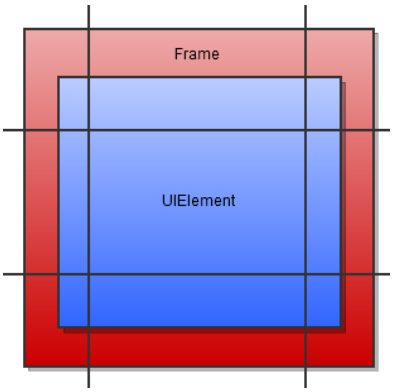

Figura 14: Adorner

Adorner está separado en nueve partes, como se aprecia en la figura 14, y cada parte tiene una funcionalidad con el *Adorner*. Por ejemplo, si el usuario realiza acciones con el ratón en la parte central superior, el *Adorner* crecerá o decrecerá verticalmente. Asimismo, las esquinas tienen doble funcionalidad ya que permiten rotar y escalar.

Para diferenciar las acciones se cambia el cursor según la acción que estemos realizando.

2.3 PROTOTIPO

Para comprobar la funcionalidad del Framework se ha desarrollado un aplicativo en forma de prototipo orientado al revelado digital. El aplicativo nos ayudará a testear el Framework y mostrar que tareas podemos realizar que sin él no podríamos.

El prototipo de demostración cuenta con cuatro módulos o ventanas:

- *Sources Manage,* herramienta que carga fotografías desde Internet o el PC.
- *Workspace*, módulo que muestra todas las imágenes cargadas.
- *Image Editor,* editor de fotografías que permite dibujar sobre la fotografía y añadir adornos.
- *Collage Creator,* permite unir varias fotografías en una sola.

Estos 4 módulos se cargan sobre un módulo principal llamado *Kernel*, que se dedica a gestionar al resto.

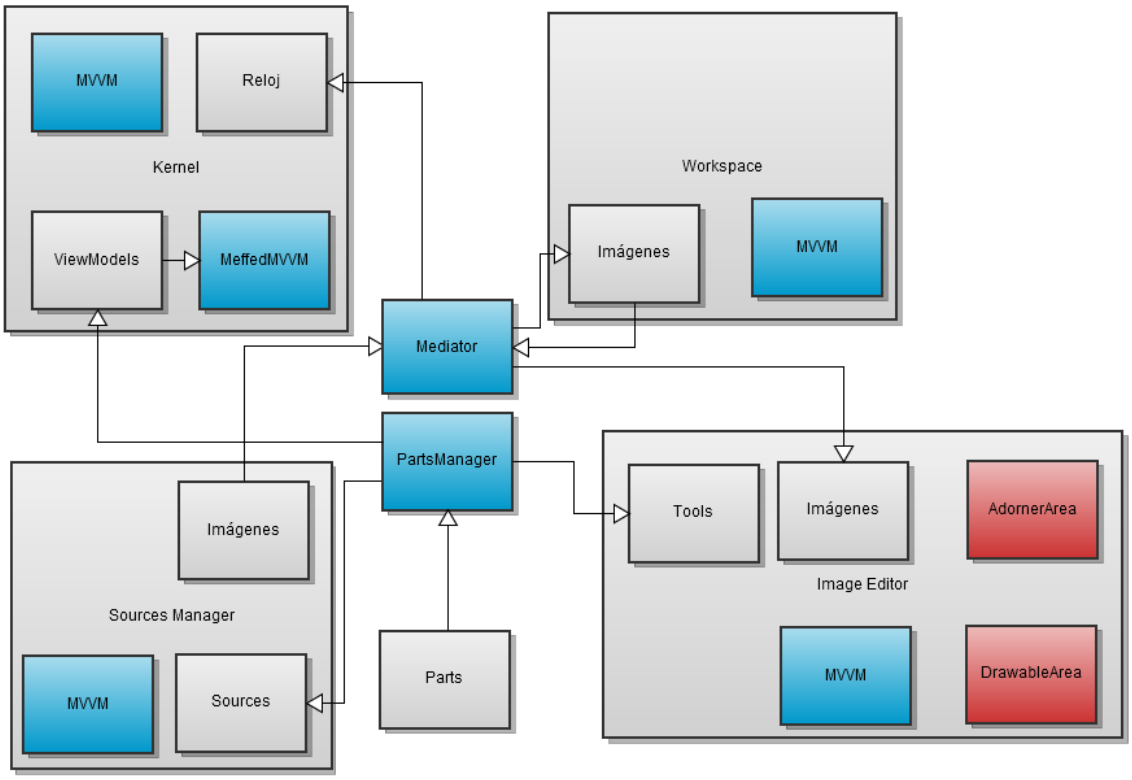

Figura 15: Aplicativo de demostración

Como vemos en la figura 15, *MVVM* está presente en todos los módulos ya que es el encargado de definir como se debe crear la ventana. Por otro lado, *Mediator* y *PartsManager* son componentes externos, y son los módulos los que solicitan información o acciones.

Mediator se utiliza principalmente para distribuir las fotografías entre los diferentes módulos, actualizar el reloj de *Kernel.*

Kernel es el único componente que requiere de *MeffedMVVM* ya que es el único que muestra ventanas.

PartsManager, además de su partición en *MeffedMVVM,* obtiene los *Sources* para *SourcesManager* y las *Tools* para *ImageEditor*.

2.3.1 KERNEL

El módulo *Kernel* (figura 16) que implementa el módulo *MVVM* (es decir tiene *View*, *ViewModel* y *Model*) es el encargado de gestionar el resto de ventanas. Es el único módulo que se crea al iniciar el aplicativo y una vez creado utiliza *PartsManager* para obtener el resto de módulos.

Kernel tiene la lógica de obtener el resto de ventanas usando el módulo *MeffedMVVM*, encargase de mostrar en pantalla el resto de módulos y comprobar si se pueden mostrar o no.

Como existen módulos que pueden realizar procesos de cálculos pesados, *Kernel* incorpora un componente que permite mostrar una barra de estado y un texto para notificar al usuario del proceso. Este componente se va actualizando gracias al *Mediator*.

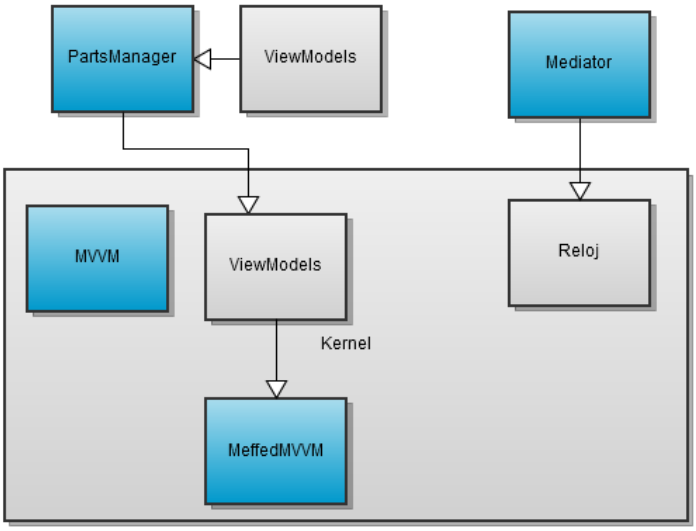

Figura 16: Kernel

2.3.2 WORKSPACE

Es el componente encargado de gestionar las fotografías (Figura 17). Implementa el patrón *MVVM* y es un claro ejemplo de la funcionalidad de *Mediator*.

El usuario puede añadir, eliminar y guardar fotografías en la sesión, y estos cambios se tienen que reflejar en el resto de módulos. Más específicamente, las tareas de *Workspace* son:

- Añadir nuevas fotografías al repositorio.
- Notificar y enviar al resto de componentes las nuevas fotografías.
- Enviar todo el repositorio de imágenes a un componente.

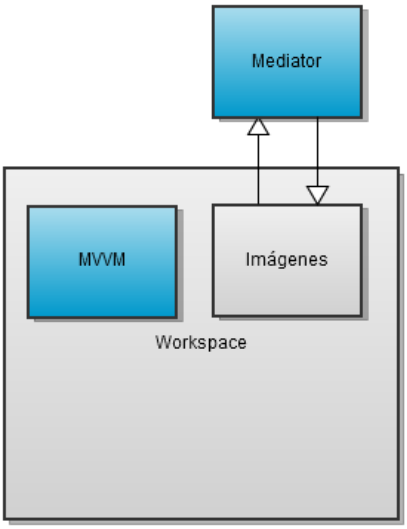

Figura 17: Workspace

2.3.3 SOURCES MANAGER

Es la herramienta para cargar fotografías (Figura 18). Su función es usar módulos para obtener imágenes como Mi PC, Facebook, Flickr…

Este ViewModel consulta la base de datos y recibe que módulos debe componer *PartsManager*. Se muestran en un listado y una vez el usuario selecciona uno, empieza la lógica del módulo.

Las imágenes seleccionadas se envían mediante *Mediator* a *Workspace*.

2.3.4 IMAGE EDITOR

Este componente (Figura 19), basado en *MVVM,* sirve para editar fotografías. Compone herramientas para la edición fotográfica desde la base de datos mediante *PartsManager*. Tiene una instancia de los dos controles creados: *DrawableArea* y *AdornerArea.*

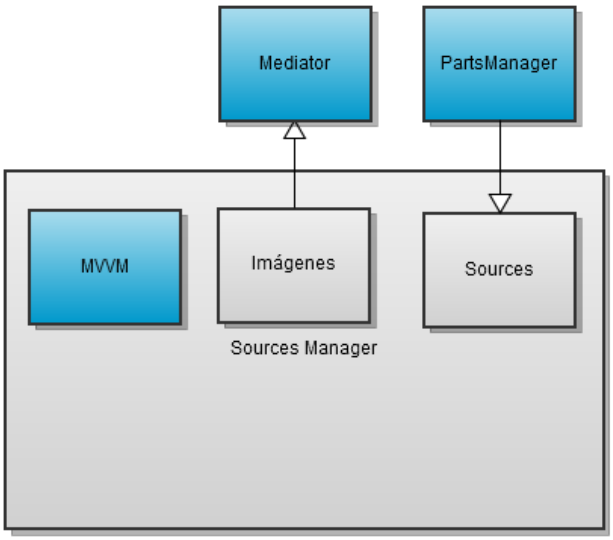

Figura 18: Sources Manager

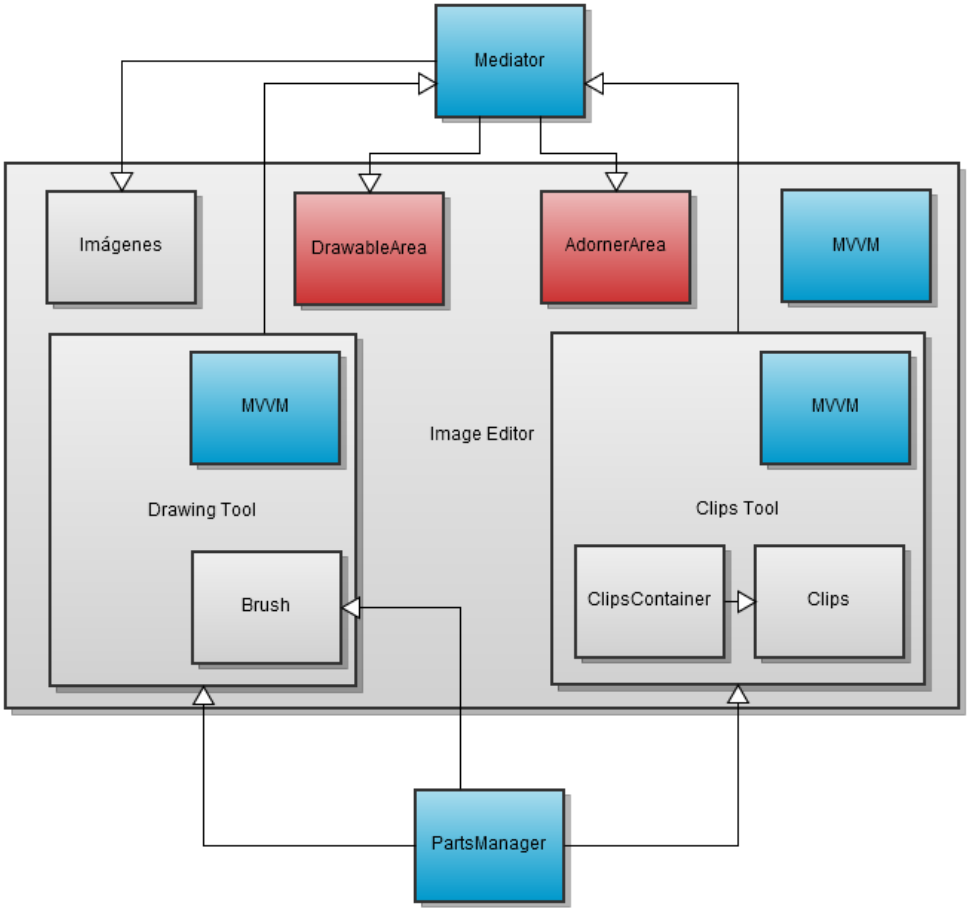

Figura 19: Image Editor

A) DRAWING TOOL

Drawing Tool es una herramienta de Image Editor que también implementa *MVVM*. Su función es la de dibujar sobre una imagen y para ello usa el control *DrawableArea*.

Como ya explicamos en *DrawableArea*, ésta requiere de objetos *Brush* para el dibujar el trazo. Para conseguirlo *Drawing Tool* realiza una consulta a la base de datos y compone los partes que contienen *Brush* mediante *PartsManager*.

Estos *Brush* se muestran en la *View* de *Drawing Tool* y cuando uno de ellos es seleccionado se envía mediante Mediator a la *DrawableArea*.

B) CLIPS TOOL

Clips Tool es una herramienta que permite decorar una imagen con un conjunto de *Clips*. Implementa *MVVM* y utiliza el *Mediator* para enviar el clip seleccionado al *AdornerArea* de *Image Editor*.

Los clips están ordenados por temáticas y cada temática tiene un fichero XML que guarda la dirección dónde están y sus nombres.

Para facilitar el manejo de los Clips se ha desarrollado un componente llamado *ClipsContainer* que ofrece estas funcionalidades:

- Descargar clips
- Guardar clips por temáticas
- Obtener clips dada una temática
- Obtener todas las temáticas

2.3.5 COLLAGE CREATOR

Este último componente (Figura 20) se ha diseñado con la finalidad de demostrar lo fácil que es crear Collages con el Framework junto al control *AdornerArea*. Implementa *MVVM* y como hemos mencionado, tiene una instancia de *AdornerArea.*

En este caso las imágenes seleccionadas por el usuario se envían al *AdornerArea.* Las imágenes están dentro de un *Adorner* al que se le puede modificar el color del borde, grosor y radio.

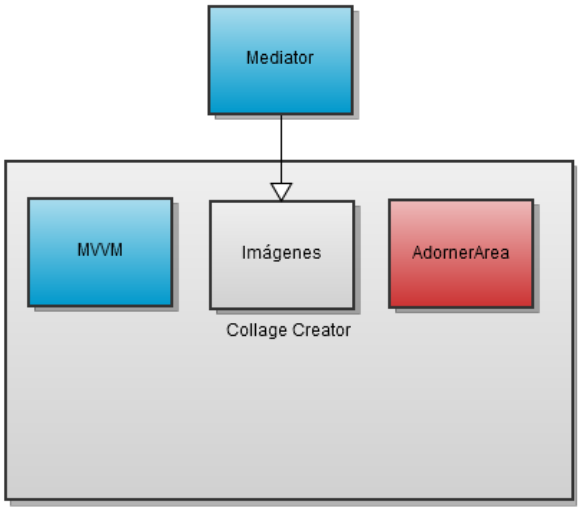

Figura 20: Collage Creator

3. RESULTADOS

En ese apartado vamos a analizar el resultado del prototipo del Framework mediante capturas de ventanas y comentarios. Analizaremos los componentes mencionados en el apartado Prototipo de Framework.

3.1 KERNEL

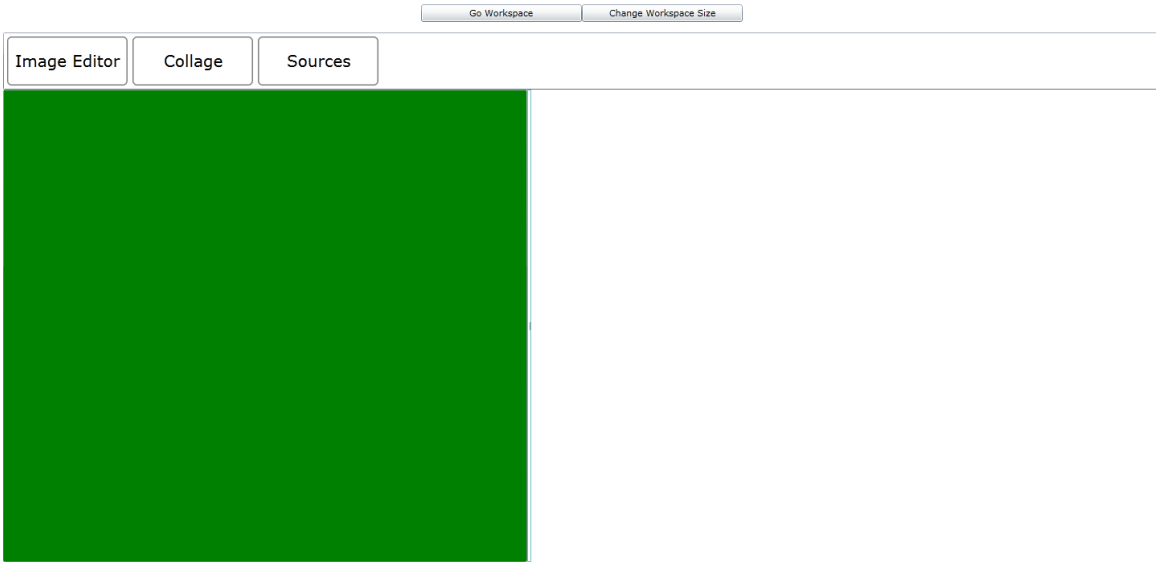

Figura 21: View de Kernel

En la ventana de Kernel (Figura 21) se puede diferenciar 3 partes, la superior donde hay botones para gestionar el Workspace, la central donde están los botones al resto de módulos y la inferior, dividida por la mitad, la parte izquierda muestra Workspace y la derecha el resto de módulos.

Gracias al desglose de la View con el Modelo que nos proporciona *MVVM* podemos:

- Mostrar por pantalla el mismo objeto tantas veces como queramos.
- Mostrar el mismo objeto de la misma forma. (Misma View, en la Figura 22)
- Mostrar el mismo objeto de manera distinta. (Diferentes View, en la Figura 23)
- Cambiar la manera en que se muestra sin alterar el modelo de datos.

3.2 SOURCES

Al seleccionar uno de los módulos, como se puede apreciar en la Figura 24, vemos como se actualiza el reloj en la esquina superior gracias a Mediator.

Una vez cargado Sources Manager, se van componiendo los Sources y desaparece el reloj, dando como resultado la Figura 25.

Las imágenes se van cargando en una lista de SoucesManager y el botón para añadir, hasta ahora deshabilitado, ahora se ha activado (Figura 26). Esto pasa gracias a *ICommand*, del componente *MVVM*, gracias a él además de la acción podemos asociar al *ICommand* una condición que verifique si se puede ejecutar la acción.

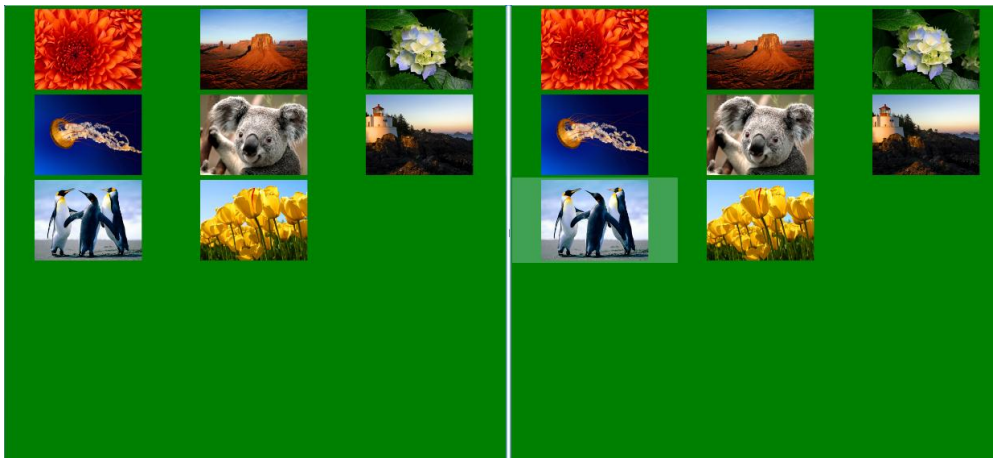

Figura 22: Mostrando Workspace dos veces con la misma View.

Figura 23: Mostrando Workspace dos veces con diferente View.

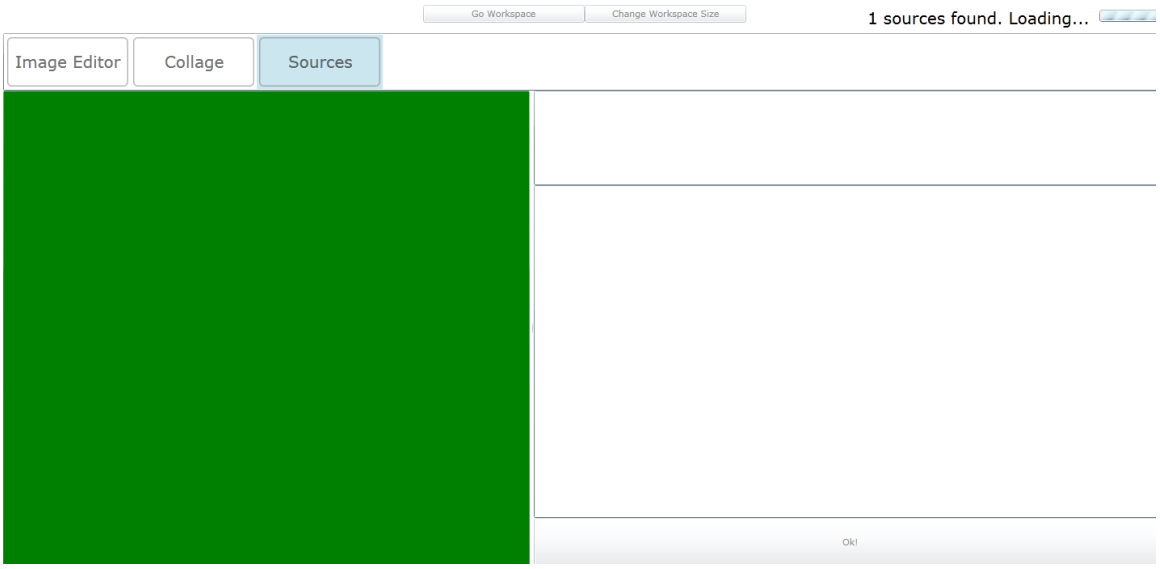

Figura 24: El reloj se actualiza.

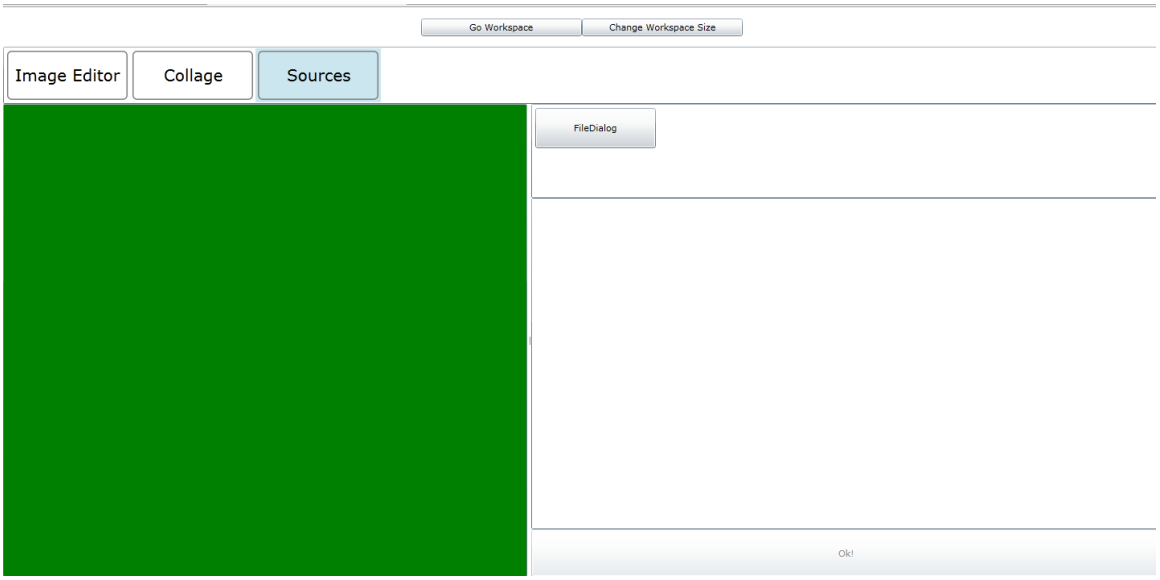

Figura 25: El reloj ha desaparecido tras cargar un módulo.

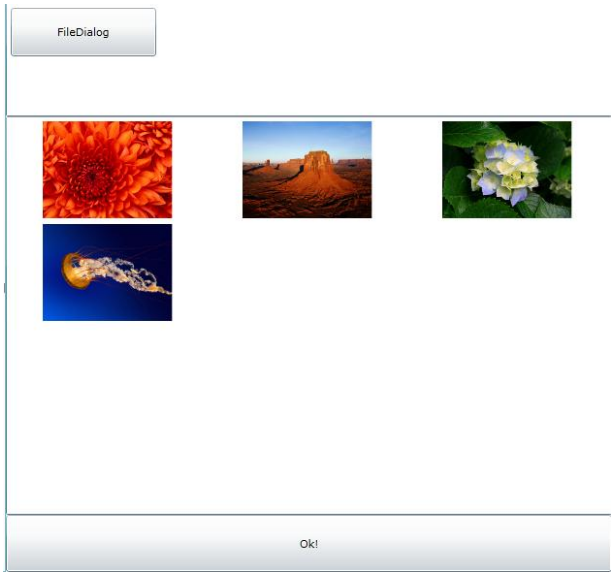

Figura 26: Sources Manager con imágenes y añadir activado.

3.3 IMAGE EDITOR

Una vez añadimos las imágenes, éstas se envían mediante *Mediator* a *Workspace* y desde éste al resto de componentes: *Image Editor y Collage*.

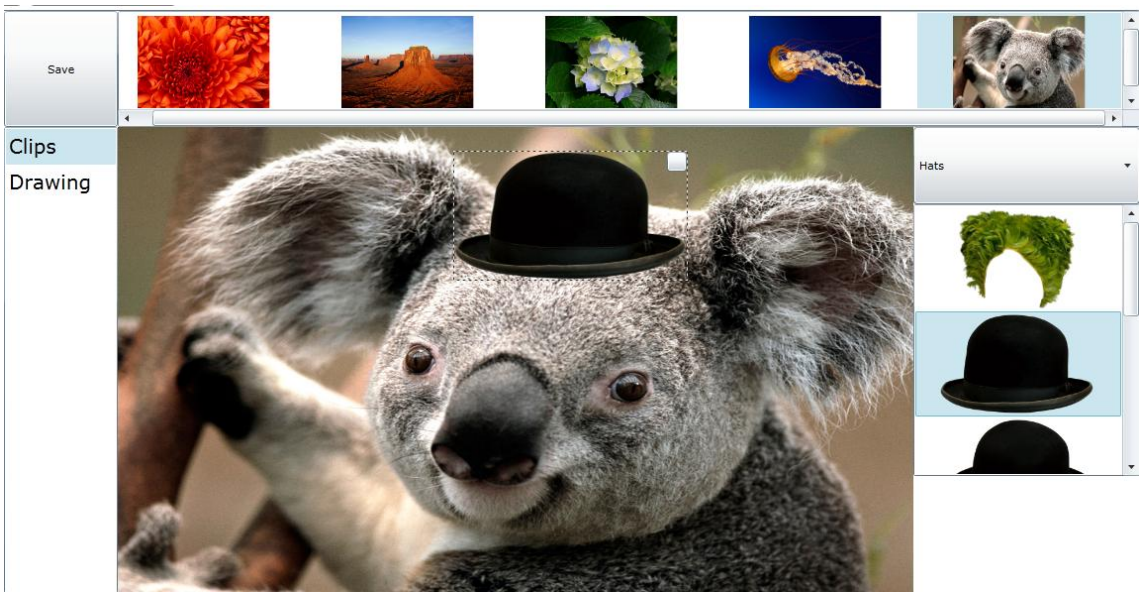

Figura 27: Image Editor con Clips Tool

En la parte superior del módulo tenemos una copia de las imágenes de *WorkSpace* enviadas a través de *Mediator*, en el lado izquierda aparecen las herramientas para editar y en el derecho se abren las opciones de cada herramienta.

Como podemos ver en la figura 27, *Clips Tool* se dibuja con un *ComboBox* con la temática, y justo debajo los clips de dicha temática.

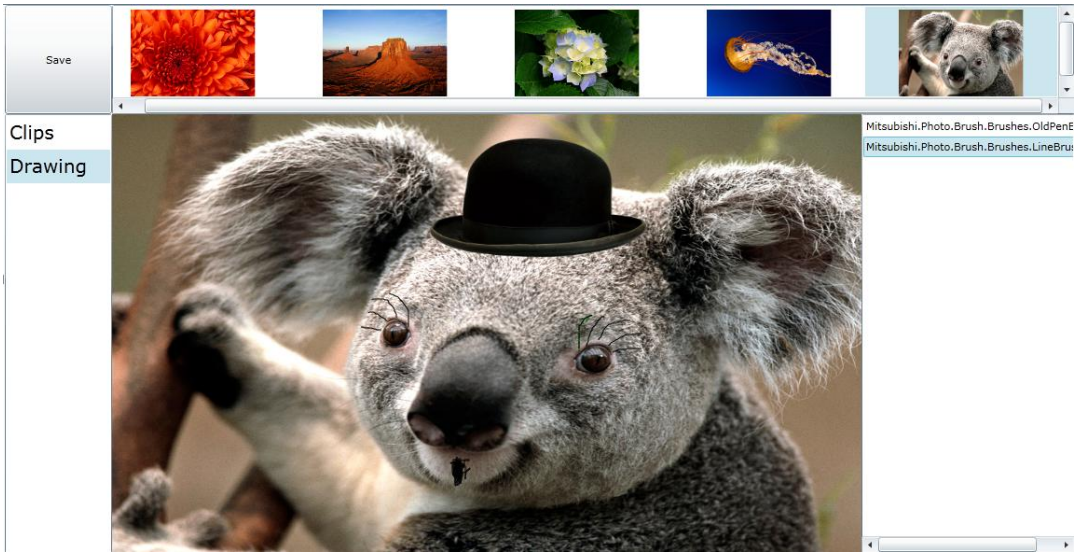

Figura 28: Image Editor con Drawing Tool

En cambio, *Drawing Tool* (figura 28) se dibuja con el listado de lápices disponibles y al seleccionar el que quieres ya puedes utilizarlo sobre la imagen.

Al componente *Image Editor* se le añadido la opción para guardar la imagen.

3.4 COLLAGE CREATOR

El último componente (figura 29 y 30) *Collage Creator*, permite añadir tantas imágenes con su *AdornerObject* como queramos y podemos cambiar las preferencias visuales de éstos con el panel de opciones de la derecha.

Figura 29: Collage Creator

Figura 30: Collage Creator

4. CONCLUSIONES Y MEJORAS

A grandes rasgos, se ha diseñado un Framework para uno de los RIA más importantes actualmente, Silverlight, que aporta a las aplicaciones modularidad y escalabilidad, también se han desarrollado dos controles que permiten editar imágenes, y por último, se ha desarrollado un entorno de prueba o prototipo para tanto el Framework como los controles.

Después de haber realizado el proyecto, el conjunto de módulos y haberlos probado, podemos extraer unas cuantas conclusiones y hacer un poco de autocrítica respecto los resultados obtenidos y los objetivos iniciales. Por lo tanto, podemos concluir:

- Silverlight es una opción muy interesante como RIA gracias al marco de .NET, que aporta un conjunto de librerías profesionales y una comunidad muy activa. Relacionado con esto, mencionar que C# ha sido un fuerte aliado
- Gracias a *MVVM* hemos conseguido desglosar el modelo de datos y que no se mezcle con la interfaz del usuario. Además de un desarrollo más efectivo permitiendo desarrollar por un lado la interfaz de usuario y por el otro el modelo de datos.
- Con la participación de *MEF* en *PartsManager* los módulos de la aplicación se cargan bajo demanda, permitiendo una mejor gestión de memoria y tener una aplicación final más modular y escalable.
- El modulo *Mediator* del framework resulta muy útil para notificar cambios pero sobre todo poder enviar objetos aporta una funcionalidad muy versátil a la hora de la comunicación entre componentes. Además su uso no implica ningún problema al rendimiento, gracias a *WeakReference* (MSDN, 2010).
- La primera versión del módulo *MeffedMVVM* ha dado un resultado muy interesante y muy funcional. En futuras versiones hay que investigar que más nos puede aportar la unión de estas dos tecnologías.
- En una primera aproximación a Silverlight para desarrollar controles para la edición digital, como *AdornerArea* o *DrawableArea,* vemos que no existe ningún tipo de limitación, pero sería interesante investigar librerías de externos.
- El framework, sin contar los controles, se puede aplicar a cualquier tipo de aplicación Silverlight. Todo esto gracias a la versatilidad, escalabilidad y modulación que aportan los módulos de *MVVM*, *PartsManager*, *MeffedMVVM* y *Mediator.*

A partir de ahora se seguirán investigando nuevas opciones y diseños para el framework a partir de las especificaciones impuestas por la empresa, como por ejemplo:

- La posibilidad de implementar un mediator por módulo o separar los mensajes y sus acciones al módulo donde apuntan. Esto permitiría más modulación aún y mejor gestión de memoria.
- Implementación de diferentes Sources online como Facebook, Flickr o Picasa.

- Investigación de otros campos relaciones con .NET como DeepZoom (DeepZoom, 2010), Accusoft Pegasus Silverlight SDK (Accusoft, 2010), WIA Services(WIA, 2010)…
- Investigar si es posible mejorar la experiencia del programador con el diseñador con *MVVM.* Ahora si un diseñador, con conocimiento de XAML, realiza toda la View del MVVM, tras él el programador tiene que modificar la View y realizar los enlaces con el ViewModel. Sería interesante que esta última tarea no hiciera falta.

Obviamente, este listado de mejoras irá en aumento ya que el Framework está en su primera versión y permite añadir módulos y realizar cambios con facilidad.

De manera autocrítica, el desarrollo de este proyecto ha requerido un esfuerzo muy grande pero los resultados obtenidos son muy positivos. Tanto para la empresa para la cual se realiza, como a nivel personal ya que me ha permitido aprender temas muy interesantes y actuales de la programación orientada a objetos. Los resultados creemos que son buenos y que, con tiempo y más gente involucrada, se puede conseguir un framework potente que de aplicaciones profesionales.

5. BIBLIOGRAFÍA Y REFERENCIAS

Abrams, Brad. 2009. MSDN Blogs: Noticia de la publicación de MEF Preview 6. . [En línea] 13 de Julio de 2009. [Citado el: 24 de Agosto de 2010.] http://blogs.msdn.com/b/brada/archive/2009/07/13/managed-extensibility-framework-mef-preview-6-silverlight-support-and-much-more.aspx.

Accusoft. 2010. Empresa dedicada al desarrollo de SDKs para .NET. [En línea] 6 de Septiembre de 2010. http://www.accusoft.com/.

Adobe. 2010. Adobe Rich Internet Applications. [En línea] 20 de Agosto de 2010. http://www.adobe.com/resources/business/rich_internet_apps.

Bishop, Judith. 2007. *C# 3.0 Design Patterns.* s.l. : O'Reilly Media, 2007.

Block, Glenn. 2010. Artículo sobre PRISM y MEF en el blog del encargado de MEF. [En línea] 25 de Agosto de 2010. http://www.sparklingclient.com/mef-and-prism/.

DeepZoom. 2010. Página en MSDN de esta tecnología .NET que permite ir mejorando la preview de una imagen según el tiempo y el ancho de banda. [En línea] 6 de Septiembre de 2010. http://msdn.microsoft.com/es-es/library/cc645050%28VS.95%29.aspx.

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley Professional, 1994.

ExpressionStudio. 2010. Página oficial de Expression Studio de Microsoft. [En línea] 21 de Agosto de 2010. http://www.microsoft.com/expression/.

GoogleMaps. 2010. Es un servidor de aplicaciones de mapas en la Web. [En línea] 20 de Agosto de 2010. http://maps.google.es/.

JavaFX. 2010. RIA de Java. [En línea] 20 de Agosto de 2010. http://javafx.com/.

Mayo, Joseph. 2001. *C# Unleashed.* s.l. : Sams, 2001.

MEF. 2010. Página oficial de Managed Extensibity Framework. [En línea] 21 de Agosto de 2010. http://mef.codeplex.com/.

Mitsubishi. 2010. Información relacionada con la empresa. [En línea] 8 de Septiembre de 2010. http://www.mitsubishi-imaging.com/photo/.

MSDN Silverlight. 2010. Información de la API de Silverlight. [En línea] 21 de Agosto de 2010. http://msdn.microsoft.com/es-es/silverlight/default.aspx.

NET. 2010. Información sobre el framework .NET de Microsoft. [En línea] 21 de Agosto de 2010. http://www.microsoft.com/net/.

Oikoumene. 2010. Explicación de RIA en castellano. [En línea] 20 de Agosto de 2010. http://oikoumene.nireblog.com/post/2010/04/14/rich-internet-applications-gwt.

PRISM. 2010. Web oficial de Prism. [En línea] 25 de Agosto de 2010. http://compositewpf.codeplex.com/.

Silverlight. 2010. RIA de Microsoft. [En línea] 20 de Agosto de 2010. http://www.silverlight.net.

Smith, Josh. 2010. MSDN Design Patterns. [En línea] 25 de Agosto de 2010. http://msdn.microsoft.com/en-us/magazine/dd419663.aspx.

Statowl. 2010. Estadísticas del uso de Internet. [En línea] 20 de Agosto de 2010. http://www.statowl.com/custom_ria_market_penetration.php.

VS2010. 2010. Página oficial de Visual Studio 2010. [En línea] 21 de Agosto de 2010. http://www.microsoft.com/visualstudio/en-us.

WIA. 2010. Conjunto de servicios para desarrollar aplicaciones que se conectan a cámaras de fotos, scanner... [En línea] 6 de Septiembre de 2010. http://msdn.microsoft.com/en-us/library/ms630368%28VS.85%29.aspx.

Wikipedia. 2010. Enciclopedia en Internet. [En línea] 21 de Agosto de 2010. http://en.wikipedia.org/wiki/.

RESUMEN

Internet se ha convertido en una clara referencia tecnológica; cada vez más las empresas apuestan por desarrollar aplicaciones en la red de redes y no quieren que su presencia pase desapercibida. Las Aplicaciones Ricas en Internet (RIA) son hasta la fecha la mejor opción para desarrollar estas aplicaciones. Este proyecto trata sobre el desarrollo de un Framework (conjunto de componentes orientados a una RIA en concreto: Silverlight) para el desarrollo de aplicaciones web.

RESUM

Internet s'ha convertit en una clara referència tecnológica; cada vegada més les empreses aposten per desenvolupar aplicacions a la xarxa de xarxes i no volen que la seva presència passi desapercebuda. Les Aplicacions Riques en Internet (RIA) són fins al momento la millor opció per desenvolupar aquestes aplicacions. Aquest projecte tracta sobre el desenvolupament d'un Framework (conjunt de components orientats a una RIA en concret: Silverlight) per al desenvolupament d'aplicacions web.

ABSTRACT

Internet has become a clear technological reference. More and more companies bet on developing internet applications and do not want its presence to pass unobserved. The Applications Rich in Internet (RIA) are the best option so far to develop these applications. This project is about the development of a Framework (set of components of a particular RIA: Silverlight) for the development of web applications.