LiveUpdate: Gestor de descargas

Memoria del proyecto

de Ingeniería Técnica en

Informática de Sistemas

realizado por

Alejandro Arroyo López

y dirigido por

Gonzalo Vera Rodríguez

Escuela de Ingeniería

Sabadell, Junio de 2011

El abajo firmante, Gonzalo Vera Rodríguez,

profesor de la escuela de ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria ha sido realizado bajo su dirección por

Alejandro Arroyo López

Y para que conste firma la presente.
Sabadell, Junio de 2011

Firmado: Gonzalo Vera Rodríguez

HOJA DE RESUMEN – PROYECTO FINAL DE CARRERA DE LA ESCUELA DE INGENIERÍA

Título del projecte: LiveUpdate:Gestor de descargas	
Autor: Alejandro Arroyo López	**Fecha:** Junio 2011
Tutor: Gonzalo Vera Rodríguez	
Titulación: Ingeniería Técnica en Informática de Sistemas	
Palabras clave • Català: WPF, C#, Web, multi idioma. • Castellano: WPF, C#, Web, multi idioma. • English: WPF, C#, Web, multi language.	
Resumen del proyecto • **Català:** La funció principal de LiveUpdate serà la descàrrega d'actualitzacions mitjançant un servidor remot permetent tenir un control integral de totes les descàrregues disponibles. El desenvolupament del projecte estarà marcat per la tecnologia en que es basa, *Windows Presentation Foundation (WPF)* per la creació d'interfícies gràfiques enriquides, i el llenguatge en que es sustenta, C#. La metodologia utilitzada estarà caracteritzada pel model Model-Vista-VistaModel (MVVM) i els estàndards corporatius que Mitsubishi aplica al seu software. Finalment, el sistema també disposarà de suport multiidioma podent visualitzar idiomes amb caracters no llatins com el rús (cirilic) o el japonès (katakana, hiragana). • **Castellano:** La función principal de LiveUpdate será la de descargar actualizaciones mediante un servidor remoto permitiendo tener un control integral de todas las descargas disponibles. El desarrollo del proyecto estará marcado por la tecnología en la que se basa, *Windows Presentation Foundation (WPF)* para la creación de interfaces gráficas enriquecidas, y el lenguaje en que se sustenta, C#. La metodología utilizada estará caracterizada por el modelo Modelo-Vista-VistaModelo (MVVM) y los estándares corporativos que Mitsubishi aplica en su software. Finalmente, el sistema también dispondrá de soporte multiidioma pudiendo visualizar idiomas con caracteres no latinos como el ruso (cirílico) o el japonés (katakana, hiragana). • **English:** LiveUpdate main function will be downloading updates through a remote server allowing you to fully control all available downloads. Project development will be based on Windows Presentation Foundation (WPF) technology that is used for creating rich graphical interfaces in C #. The methodology will be characterized by the Model-View-ViewModel (MVVM) pattern and Mitsubishi corporate standards applied in their software. Finally, the system will also have multi-language support to display non-Latin character languages such as Russian (Cyrillic) or Japanese (Katakana, Hiragana.)	

CONTENIDO

1 INTRODUCCIÓN

El siguiente documento pretende sintetizar el trabajo realizado durante el Proyecto de Final de Carrera (PFC) de la Ingeniería Técnica en Informática de Sistemas cursada en la Escuela de Informática de Sabadell de la UAB.

LiveUpdate formará parte del nuevo software de impresión fotográfica, *Kiosk Gifts*. *LiveUpdate* ha sido escogido como proyecto final de carrera por una combinación de factores tanto personales como profesionales; la empresa deseaba desarrollar un nuevo sistema de actualización de software para nuestros equipos y yo estaba interesado en el desarrollo de una solución para el departamento de fotografía.

Dentro del contexto de la aplicación *Kiosk Gift,* el ámbito concreto que incluye *LiveUpdate* es el desarrollo del módulo de gestión de descarga de actualizaciones y parches combinado con la implementación de un sistema de obtención de información distribuido en un servidor. La diferencia básica entre los dos componentes es que mientras *LiveUpdate* se encarga de gestionar todas las tareas relacionadas con la descarga y gestión de las actualizaciones (descargar, pausar, retomar), el sistema de obtención de información distribuido solo recoge la información de las descargas para que *LiveUpdate* pueda gestionarla.

LiveUpdate proporcionará una herramienta potente y funcional para la gestión de actualizaciones y parches, aportando más valor añadido al área de soporte de equipos con sistema *Kiosk Gifts*. De este modo el usuario final siempre dispondrá de las últimas mejoras del sistema, así como de nuevos productos en mercado.

En la siguiente figura podemos ver una captura de la pantalla principal de *Kiosk Gifts.*

Figura 1.1: Pantalla principal de *Kiosk Gifts* (Software de impresión fotográfica).

El PFC se desarrolla bajo la supervisión de la empresa Mitsubishi Electric B.V Spain. En ningún momento se realizará un convenio de trabajo debido a que el PFC se realizará a modo personal (fuera de horario laboral, en la misma empresa), como propuesta de un desarrollo que la empresa necesita en el departamento de I+D del cual formo parte.

En los siguientes apartados se realiza una exposición ampliada la empresa, revisando los antecedentes de la compañía, así como la localización donde se ha realizado este proyecto y finalmente del entorno de trabajo.

1.1 La empresa

Mitsubishi es una de las mayores compañías de Japón. Fue fundada el 13 de mayo de 1870 por Yataro Iwasaki, hijo de una familia samurái. Desempeñó un importante papel en la transformación de Japón en una sociedad industrializada. Se dedicó en un principio al transporte marítimo. La marca y el nombre de Mitsubishi, se refieren a 'tres diamantes'. En la actualidad, es un consorcio de compañías descentralizadas.

A finales del siglo XIX, la compañía, que genera ella sola la mitad del tráfico marítimo japonés, inicia un proceso de diversificación que finalizaría con la creación de tres entidades:

- **Mitsubishi Bank**, banco fundado en 1919. Tras fusionarse en 1996 con el Banco de Tokyo, el grupo se ha convertido en el primer banco de Japón.

- **Mitsubishi Corporation**, fundada en 1893, sirve a la financiación interna del grupo.
- **Mitsubishi Heavy Industries** engloba las actividades industriales del grupo (Mitsubishi Electric, Motors, Atomic Industry, Chemical y Precision).

Como curiosidad, Mitsubishi había producido material militar para el ejército japonés durante la segunda guerra mundial, incluido el famoso caza Zero (Fig. 1.3) de los pilotos kamikaze.

Figura 1.3: Caza Zero (de producción Mitsubishi).

Todas las operaciones de España, Portugal y América latina se manejan desde las oficinas de Mitsubishi Electric en Sant Cugat del Vallès (Fig. 1.2). Es ahí donde se realiza el desarrollo a nivel mundial de productos relacionados con la fotografía.

Figura 1.2: Centro de I+D de Mitsubishi Electric (Sant Cugat).

1.2 Entorno de trabajo

Investigación y desarrollo (I+D) es el departamento dedicado, entre otras funciones, al desarrollo de software, el cual estará dividido en dos unidades. La primera es I+D-Photo donde se desarrollan soluciones para sistemas de Fotografía Digital (como podría ser software para kioscos digitales). La segunda es I+D-Secu donde se implementan soluciones de video vigilancia y seguridad para cámaras IP y video grabadores.

Una vez situados en el ámbito de I+D-Photo, observamos que se realizan simultáneamente diferentes proyectos, tanto a nivel estatal como mundial, incluyendo *Kiosk Gifts* que es un conjunto de aplicaciones para impresión fotográfica (Fig. 1.4) que incluye: Impresión fotográfica, creación de calendarios y felicitaciones, Fotocarnet, Instant photo álbum y mosaicos fotográficos.

Figura 1.4: Productos Mitsubishi (Kiosco, instant photo álbum y Easy Calendar).

El equipo de trabajo utiliza como herramienta de desarrollo WPF, y como lenguaje de programación C#.

WPF es una tecnología desarrollada por Microsoft que permite potenciar las capacidades de desarrollo de interfaces de interacción integrando y ampliando las mejores características de las aplicaciones Windows y de las aplicaciones web. Podemos ver un gráfico (Fig. 1.5) con toda la funcionalidad de WPF.

Figura 1.5: Tecnología WPF.

1.3 Motivaciones

El hecho de cursar una Ingeniería Técnica en Informática de Sistemas comporta la realización de un PFC. Es aquí donde el alumno demuestra lo que ha aprendido durante los años que han durado sus estudios, aplicando los conocimientos teórico-prácticos adquiridos, además de demostrar su capacidad de investigación y síntesis delante de un posible caso real.

El PFC comporta ciertos alicientes de cara a su realización, tales como el hecho de que una vez finalizado se habrá terminado la carrera. Este es el motivo principal, la obtención de un título que certifique que el alumno ya puede ejercer como Ingeniero Técnico, la satisfacción personal de haber llegado al final consiguiendo satisfactoriamente los hitos que se habían propuesto desde el principio.

En este caso, al ser un proyecto que se realiza como propuesta a una necesidad que tiene la empresa, comporta otras motivaciones para llegar al final, como es el hecho de sentirse realizado profesionalmente y de finalmente poder ver implantado en un cliente final, con todas las mejoras, la solución implementada.

Otro punto a tener en cuenta es la tecnología utilizada, WPF, con pocos años de vida y propia de Microsoft con la que nunca se ha trabajado y la cual ofrece un potencial enorme en cuanto a la creación de aplicaciones atractivas y funcionales para el usuario final.

1.4 Objetivos

Los objetivos son ganar cuota en el mercado, no quedarse atrás en relación a la competencia y modernizar nuestro software. Al utilizar a nuevas tecnologías, como es el caso de WPF, aumentaremos el atractivo visual de la aplicación actual, dándole un look profesional y moderno así como mejorando el carácter funcional de la misma. Es por ello que en este proyecto, mediante la incorporación de *LiveUpdate* a la aplicación *Kiosk Gifts*, se contribuye a la consecución de los objetivos generales establecidos.

Como objetivos personales tenemos la adquisición de conocimiento y experiencia a la hora de movernos en entornos de desarrollo complejos. Conocimiento de todas las fases de un proyecto, desde el primer prototipo hasta la implantación final en cliente con sus posibles errores, es decir, todo el proceso que conlleva la realización de un proyecto de grandes dimensiones.

1.5 Estructura de la memoria

La documentación del proyecto se estructura en capítulos organizando y esquematizando los puntos clave más importantes que la caracterizan.

- El primer capítulo de introducción.
- El segundo capítulo recogerá el Análisis de Viabilidad estableciendo el objetivo particular de *LiveUpdate* así como la planificación temporal del mismo, hasta concluir si es el proyecto viable.
- El tercer capítulo tratará el Análisis, así que se realizará un estudio en profundidad de las carencias del sistema actual y se especificarán los nuevos requerimientos del sistema.
- El cuarto capítulo describirá el diseño técnico del proyecto.
- El quinto capítulo mostrará los resultados de la nueva funcionalidad dividida en módulos.
- El sexto capítulo definirá que pruebas son necesarias para la validación del resultado descrito en el punto interior además de describir como llevaremos a término las diferentes pruebas del sistema.
- Las conclusiones y mejoras se reflejarán en el séptimo punto.
- A modo de complemento bibliográfico el octavo apartado recopilará las referencias bibliográficas de consulta y otros recursos utilizados.
- Finalmente, añadiremos diferentes anexos a la documentación con información complementaria a nuestro proyecto.

2 ESTUDIO DE VIABILIDAD

2.1 Introducción

En este capítulo se expondrá el estudio previo que se ha llevado a cabo sobre la viabilidad del proyecto y todas las conclusiones extraídas.

2.1.1 Objetivo

Todo proceso de creación de software finaliza con la instalación de la aplicación en un cliente, a continuación se debe continuar manteniendo y corrigiendo errores que aparecen a medida que se usa; aun así se llega a un nivel en el que el software funciona perfectamente, ya no falla y es totalmente eficiente.

Ahora bien, llega un momento en que ya no cubre las necesidades totales del cliente, o se desea una mejora substancial en el conjunto de la aplicación, así que es necesario modernizarla.

La división de fotografía se encuentra en esta situación en cuanto al módulo de "actualización de software" del nuevo software que está desarrollando (*Kiosk Gifts*), el sistema actual tiene prácticamente 8 años y su mantenimiento y funcionalidad comienzan a ser problemáticos, ya que una simple modificación comporta una complejidad en el trabajo que no beneficia ni al cliente ni a la empresa.

Existe la necesidad real, de mejorar el antiguo sistema de "actualización de software" el cual se basa principalmente en la actualización de todo el software que existe en el Kiosco.

Por tanto, el objetivo sería actualizar/mejorar el sistema actual a nivel de presentación y funcionalidad y por ello se propone realizar un nuevo sistema de descargas utilizando una nueva

tecnología para conseguir un producto (*LiveUpdate*) con una interfaz más moderna y una funcionalidad más completa.

LiveUpdate es un proyecto a gran escala, el desarrollo empezará de cero y englobará tanto funcionalidad actual como nueva; siempre pensando en un producto válido para cualquier otro software que tenga la necesidad de utilizarlo, es decir, ha de ser totalmente escalable, por ello crearemos un *Framework* que podrá ser utilizado por cualquier aplicación de nuestro sistema.

El objetivo de este proyecto es mejorar la tecnológica de aplicación en su desarrollo, así como aumentar la usabilidad del software y mejorar el carácter funcional que puede ofrecer al usuario. Por ello, se analizará las diferencias con la aplicación actual posibilitando nuevas funcionalidades en el nuevo software y actualizando la plataforma de desarrollo a una herramienta más moderna.

2.1.2 Contexto

Debido a que los sistemas creados están relacionados con la fotografía digital (Fig. 2.1), es entonces, en copisterías, kioscos, centros comerciales o tiendas de fotografía profesional donde se encuentra en usuario final de la aplicación así como el propio cuerpo técnico de Mitsubishi (*Support*), encargado de la instalación y puesta a punto de los equipo en el cliente final.

Figura 2.1: Kiosco Mitsubishi con mueble de exposición DPS Kiosk.

El perfil de los usuarios finales (Fig. 2.2) que utilizan nuestros sistemas es muy variado, por ello, el kiosco ha de ser lo más intuitivo y sencillo de utilizar posible.

Figura 2.2: Usuario final utilizando equipo Kiosco.

2.1.3 Fuentes de información

Para la realización de este proyecto es necesaria una documentación, fiable, clara y concisa, donde queden plasmados todos los detalles, ya que, a la hora de desarrollar todo el proceso, este será más fácil, transparente y ordenado.

Las fuentes de información son:

Información funcional: es aquella que indica que debe hacer el software, a quien va destinado y que funciones debe realizar. Esta información puede venir de diferentes perfiles profesionales, desde responsables de distribuidores de sistemas fotográficos hasta clientes finales que ya disponen de un kiosco en su establecimiento y conocen profundamente el producto actual.

En primer lugar, el cliente dice que quiere, como lo quiere y que ha de hacer. Su visión es el primer contacto con el que será en nuevo software, una definición explicada de forma muy informal, pero útil, ya que la persona que aporta su visión será a la vez un usuario final, quien realmente debe usar la aplicación.

La información funcional también la puede sugerir el jefe de proyecto, el jefe de marketing o el propio analista. Esta información vendrá dada de una manera más técnica que la obtenida del propio cliente, y por política de empresa, será todo por escrito.

Reuniones con el director del proyecto o el arquitecto para concretar puntos a resolver o posibles dudas encontradas en la documentación. Estas reuniones sería interesante que se hicieran cada 2 semanas para poner en común que se ha hecho, además de corregir posibles errores de requerimientos o proponer alternativas al diseño o codificación.

Información técnica: es aquella que indica cómo se debe realizar el proyecto, de qué manera, y que herramientas y metodologías existen o se pueden usar para realizarlo.

Esta información se obtiene del responsable técnico del proyecto. Permite resolver dudas de cómo usar la arquitectura escogida, posibles opciones o maneras de hacerlo, ya que suele ser quien conoce toda la parte técnica de la metodología de trabajo y quién mejor puede aconsejar a la hora de desarrollar, siempre técnicamente hablando.

Otra opción es la documentación que ofrece Internet. En este caso se dispone de la página web de msdn (http://msdn.microsoft.com/es-es/default). Aunque WPF es una tecnología con poco tiempo de vida, ya existe documentación extensa en la que apoyarnos al 100% para poder resolver dudas. Para poder visualizar aplicaciones de ejemplo también se puede acceder a la web de codeproject (www.codeproject.com).

2.2 Objeto

LiveUpdate debe disponer de una interfaz de usuario con un diseño atractivo y sencillo de utilizar, debido a que la horquilla de usuarios que disponen de nuestros equipos incluye desde usuarios con pocos conocimientos hasta expertos en informática. Además deberá informarnos del estado de la descarga y permitirnos proseguir con la misma en caso de fallo o pérdida de conexión, todo esto un modo automático y transparente para el usuario.

También se pretende la gestión integral de la instalación del software una vez este descargado, además de la instalación del software descargado, se podrá gestionar la parada y arranque de otras aplicaciones del sistema, incluyendo el reinicio total del equipo (Kiosco).

Para minimizar los fallos críticos que puedan aparecer en *Kiosk Gifts*, también se implementará un sistema de descarga automática (*Mode Silent*) para el cual se comprobará diariamente si existen descargas críticas y en caso de existir se pondrán a la cola de las descargas (sin ser estas visibles por el usuario), se descargarán e instalarán de modo completamente transparente. Para no penalizar las descargas seleccionadas por el usuario es posible modificar la proporción de descarga modificando el fichero XML de configuración por parte del operador.

Debido a que *Kiosk Gifts* es un software de implantación mundial, será necesario implementar en LiveUpdate un módulo de localización que permita la traducción del software por parte de cualquier filial de Mitsubishi y un sistema de soporte de *UNICODE* para lenguajes como el japonés o el ruso (ejemplo de caracteres japoneses: 売春婦 y rusos: дерьмо).

Otra de las mejoras que nos aporta la tecnología que utilizaremos será la posibilidad de personalizar la interfaz de usuario, dando a elegir a cada usuario la experiencia de pantalla que más se adapte a su gusto.

El modelo de desarrollo es un modelo llamado MVVM (Fig. 2.3), podemos encontrar referencias del mismo en *http://msdn.microsoft.com/es-es/magazine/dd419663.aspx*.

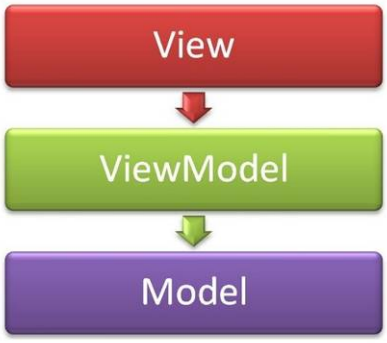

Figura 2.3: Diagrama simplificado de patrón MVVM.

Toda la estructura de la aplicación estará dividida en componentes siguiendo unos parámetros establecidos inicialmente. A continuación podemos ver la estructura básica (Fig. 2.4) de nuestro aplicativo:

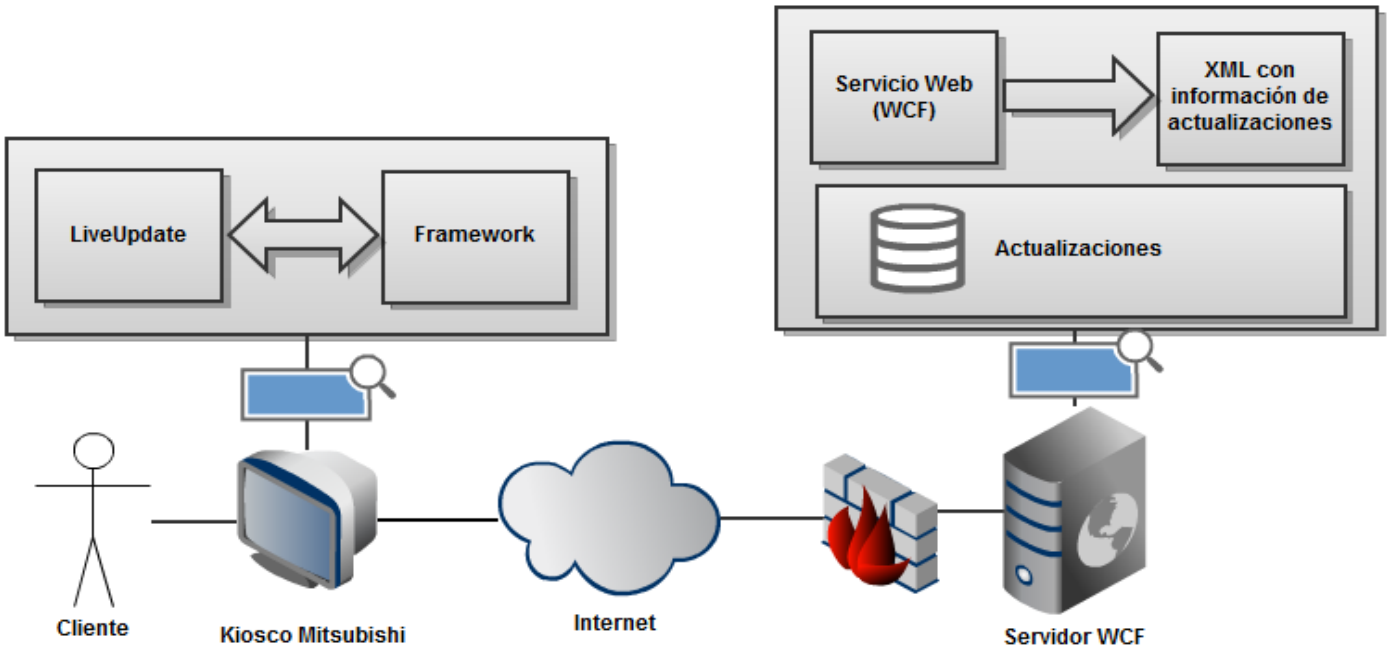

Figura 2.4: Estructura básica de LiveUpdate.

Finalmente, aunque el PFC está planificado en etapas, muchos de los trabajos podrán realizarse en paralelo debido sobre todo al diseño de componentes independientes entre si.

2.3 Posibles alternativas

Como posibles alternativas a la utilización de WPF para realizar el proyecto existen en el mercado las siguientes soluciones:

- **Flash:** con un marcado componente gráfico que lo convierte en una herramienta de desarrollo completa, para crear principalmente elementos multimedia e interactivos para Internet. (*http://es.wikipedia.org/wiki/Adobe_Flash*).
- **Java + Swing:** es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas. (*http://es.wikipedia.org/wiki/Swing_%28biblioteca_gr%C3%A1fica%29*).
- **AWT:** es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC) - la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java. (*http://es.wikipedia.org/wiki/AWT*).

Como primera opción nos hemos decantado por la opción del WPF debido a la gran potencia que puede ofrecer para la creación de controles visualmente enriquecidos así como funcionalidades 3d, además WPF cuenta con el apoyo de C# como lenguaje.

En varios de los últimos desarrollos en la compañía se ha utilizado esta tecnología obteniendo muy buenos resultados desde el punto de vista del desarrollo de aplicaciones de escritorio, con productos robustos a nivel de fallos y visualmente atractivos. Inicialmente las aplicaciones eran desarrolladas en Delphi (primeras versiones de los kioscos) y java, lo cual implicaba tener en cuenta una serie de limitaciones técnicas que dificultaban el desarrollo del proyecto, como pueden ser, la multiresolución, el soporte a Unicode o la localización. Es cuando se decide dejar de desarrollar en Delphi y parcialmente en java para pasar a WPF y C#.

Esta solución ha demostrado que se pueden generar aplicaciones potentes, solucionando problemas anteriormente complicados de un modo más sencillo. Todo esto sin penalizar el rendimiento.

Aunque existen alternativas posibles (Flash, Swing o AWT), la experiencia en otras tecnologías ha demostrado que WPF es la mejor opción para LiveUpdate ya que aporta una solución completa.

2.4 Recursos

A continuación describiremos el marco técnico del desarrollo del proyecto tanto a nivel de software como de hardware.

Software

- Para desarrollar utilizaremos Windows Vista, ya que se disponen de licencias de Windows y es uno de los sistemas operativos más extendidos actualmente.
- Para la publicación del web service y de la estructura de ficheros de descarga utilizaremos Windows Server 2008, ya que la empresa dispone de licencias y además, el sistema operativo tiene soporte nativo (IIS 7.0) para el web service.
- .Net Framework 3.5 sp1 el cual es de distribución libre (bajo condiciones de Microsoft), así que podemos redistribuirlo sin ningún problema.
- Todo el código implementado ha estado desarrollado en el entorno de programación Microsoft Visual Studio 2010, ya que también se disponen licencias para ese fin. Además, soporta numerosas versiones de Microsoft .Net Framework, eXtensible Application Markup Language (XAML) y C#, que son dos de los lenguajes de programación que se han utilizado.
- Microsoft Expression Blend 4 es un programa para la edición de interfaces. Permite además, trabajar directamente con la solución de Microsoft Visual Studio 2010.
- Microsoft Office 2010 se ha utilizado para realizar la memoria y como en los otros productos se dispone de la pertinente licencia.
- Microsoft Project ha sido utilizado para planificar temporalmente el proyecto y las tareas a realizar en cada momento. En este caso también se dispone de licencias de la aplicación.

Hardware

- 2 x Dell Intel Core 2 duo 2.53 con 3 GB de RAM.
- Sony LN1M – Intel Core 2 duo 2.53 (E7200) con 3 GB de RAM.
- Máquina para hospedaje de web service y ficheros (empresa): Máquina virtual, la cual permite asociar tantos recursos como sean necesarios (RAM, CPU, etc.).
- Máquina encargada de la redirección de IP fija a máquina virtual (IP por la que accedemos a las descargas). Datos técnicos no especificados por la empresa).
- Kiosco Mitsubishi (para demo) con Intel Core 2 duo 2.40 con 1 GB de RAM.

2.5 Planificación

El plan de proyecto es la pauta que seguiremos durante todo el proceso del PFC. Si está bien definido, el trabajo es mucho más dinámico. Para evitar posibles desvíos el proyectista deberá seguir la planificación de manera clara y ordenada.

El PFC se llevará a cabo en horas no laborales, en la propia empresa, por tanto es en este lugar donde se intentará cumplir al máximo la planificación que se ha realizado.

En principio, la idea es que se dedique alguna hora después de la jornada laboral, y utilizar horas muy puntuales para posibles problemas que puedan surgir durante el desarrollo.

De este modo, se estima que el proyecto dure aproximadamente entre 180-200 horas, que es el tiempo que estipula la propia asignatura de "Proyecto en informática de Sistemas" de la Escuela de Ingeniería del campus de Sabadell.

Para completar la planificación definiremos una serie de fases donde se detallarán los pasos a seguir dentro del desarrollo del PFC.

2.5.1 Fase de documentación

A continuación expondremos que información y que pasos serán necesarios seguir para llevar a cabo la fase de documentación del PFC:

- **Recopilación de información:** Información sobre el proyecto a desarrollar, para así poder valorar su viabilidad y conocer el estado del arte del tema a tratar.

- **Lectura de *Technical papers*:** Una vez recopilada la información es importante realizar tantas lecturas como sea necesario para poder asimilar y entender los nuevos conceptos relacionados con patrones (MVVM), lenguajes WPF, etc.

- **Familiarización con WPF y C#:** ya que el proyecto está implementado en WPF y C# se ha reservado un tiempo de familiarización con estos lenguajes.

- **Informe previo:** documento obligatorio para iniciar el proyecto y dejar constancia de que se ha estudiado la viabilidad y se ha planificado debidamente.

2.5.2 Fase de análisis

A continuación expondremos los pasos a seguir en la fase de análisis:

- **Análisis y valoración de los pasos que deben seguir para la implementación del proyecto**: Como el proyecto puede formar parte de un producto de la empresa deberá prestar especial atención a realizar la implementación lo más modular posible para el producto ya existente pueda reutilizar si se diera el caso, partes ya implementadas en el proyecto.

- **Comprensión profunda del sistema de gestión de descargas que se pretende implementar**: Se tendrán en cuenta posibles requerimientos que existen en estos momentos y posibles requerimientos que puedan aparecer en un futuro.

2.5.3 Fase de diseño

Definiremos los puntos necesarios para el diseño de nuestro sistema, para ello dividiremos el diseño en dos partes:

- **LiveUpdate:** Diseño de los componentes y lógica necesarios para la gestión de actualizaciones.

- **Framework:** Diseño de todo el conjunto de módulos necesarios para el funcionamiento de LiveUpdate, desde módulos de localización hasta controles. En framework se diseñarán todos los componentes que puedan ser reutilizados por otras aplicaciones.

2.5.4 Fase de implementación y test

A continuación definiremos los puntos necesarios para la fase de implementación y test:

- **Implementación interfaces de la aplicación:** Siguiendo patrón MVVM se implementará la estructura de la pantalla así como su funcionalidad a la hora de mostrar la información.

- **Implementación de Framework I:** Tarea reservada a la implementación de módulos que formarán parte del framework tales como gestor de descarga, manejador de XML, controles y estilos.

- **Implementación de Framework II:** Tarea reservada a la implementación de módulos que formarán parte del framework tales como la gestión de configuraciones, encriptación y gestión de ficheros.

- **Implementación de sistema de instalación de aplicaciones (Framework III):** Gestor de instalación para una vez descargadas las actualizaciones ejecutar todos los pasos necesarios en el sistema para su instalación.

- **Implementación de sistema de localización (Framework IV):** Tarea planificada para tener soporte multiidioma del software (pudiendo este modificarse en tiempo de ejecución).

- **Test:** Test sobre la aplicación creada permitirá para comprobar su correcto funcionamiento en un entorno real.

- **Valoración de resultados:** Después de analizar los resultados conoceremos si hay algún punto de la aplicación que debamos modificar. Por tanto, se deberá hacer especial hincapié en este apartado para evitar futuros fallos del sistema.

- **Reajustes y modificaciones:** Tarea destinada a realizar las modificaciones pertinentes una vez hechas las primeras valoraciones.

2.5.5 Fase de entrega

A continuación describiremos que pasos deberemos seguir para la fase de entrega:

- **Memoria escrita:** Tarea que se irá realizando durante los meses de trabajo en el proyecto para tener en soporte escrito del trabajo hecho y de los resultados obtenidos. Finalmente se deberá hacer una última revisión de este documento para finalizarlo de cara a la entrega final.

- **Exposición oral:** son días reservados para estructurar la exposición oral y para la preparación de los puntos más interesantes del proyecto a explicar.

2.5.6 Hitos

Hito 1: Tener realizada la primera versión de la interfaz, utilizando patrón de diseño MVVM. Implementar parte de la lógica de la aplicación así como algunos de los módulos del Framework.

Hito 2: Enlace de la aplicación con sistema *web service* y proseguir con módulos necesarios.

Hito 3: Tener acabado el sistema de localización de la aplicación así como el sistema de gestión de instalaciones.

Hito 4: Tener la totalidad de la aplicación realizada.

2.5.7 Reuniones con el tutor

El propósito de las reuniones será explicar cómo va avanzando el proyecto y poner en común tanto lo que está hecho como lo que está planteado hacer. Deben estar planificadas, una cada mes, para permitir apreciar la evolución del proyecto y determinar si la línea de actuación es la correcta.

2.5.8 Diagrama de Gantt

Se ha realizado un diagrama de Gantt para poder seguir un plan de actuación, distribuyendo temporalmente las tareas que se deben realizar. La mayoría de las tareas se realizarán en paralelo, alternando unas con otras su implementación. Además, se dedicarán entre 1 y 2 horas diarias su realización.

Nombre de tarea	Horas dedicadas	Comienzo	Fin
PFC	**185 horas**	**jue 21/10/10**	**sáb 30/04/11**
Fase de documentación	**13 horas**	**jue 21/10/10**	**mié 08/12/10**
Recopilación de información	3 horas	vie 22/10/10	lun 15/11/10
Lectura de Technical papers	3 horas	lun 25/10/10	mié 17/11/10
Familiarización con WPF y C#	4 horas	lun 25/10/10	jue 25/11/10
Informe previo	3 horas	mar 07/12/10	mié 08/12/10
Fase de análisis	**18 horas**	**lun 01/11/10**	**lun 29/11/10**
Análisis y valoración de los pasos que deben seguir para la implementación	12 horas	lun 01/11/10	lun 15/11/10
Comprensión profunda del sistema de gestión de descargas	6 horas	lun 15/11/10	lun 29/11/10
Fase de diseño	**30 horas**	**mié 01/12/10**	**lun 24/01/11**
Diseño de componente LiveUpdate	10 horas	mié 01/12/10	mié 15/12/10
Diseño de componentes para framework	20 horas	jue 16/12/10	lun 24/01/11
Fase de implementación y test	**90 horas**	**sáb 30/10/10**	**mié 20/04/11**
Implementación interfaces de la aplicación	10 horas	sáb 30/10/10	lun 15/11/10
Implementación de Framework I	15 horas	lun 15/11/10	jue 30/12/10
Implementación de Framework II	15 horas	mié 01/12/10	jue 06/01/11
Implementación de sistema de instalación de Framework III	10 horas	mar 04/01/11	vie 28/01/11
Implementación de sistema de localización de Framework IV	10 horas	lun 03/01/11	mié 26/01/11
Test	15 horas	lun 03/01/11	mié 20/04/11
Valoración de resultados	5 horas	lun 24/01/11	mar 29/03/11
Reajustes y modificaciones	10 horas	mar 01/03/11	mié 20/04/11
Fase de entrega	**34 horas**	**mar 07/12/10**	**sáb 30/04/11**
Exposición oral	4 horas	lun 25/04/11	sáb 30/04/11
Memoria escrita	30 horas	mar 07/12/10	sáb 30/04/11
Hito 1	1 hora	mar 07/12/10	mar 07/12/10
Hito 2	1 hora	mar 11/01/11	mar 11/01/11
Hito 3	1 hora	jue 27/01/11	jue 27/01/11
Hito 4	1 hora	mar 08/02/11	mar 08/02/11
Reuniones con el tutor PFC	5 horas	**jue 21/10/10**	**mar 08/02/11**

A continuación podemos ver el diagrama de Gantt resultante (Fig. 2.5):

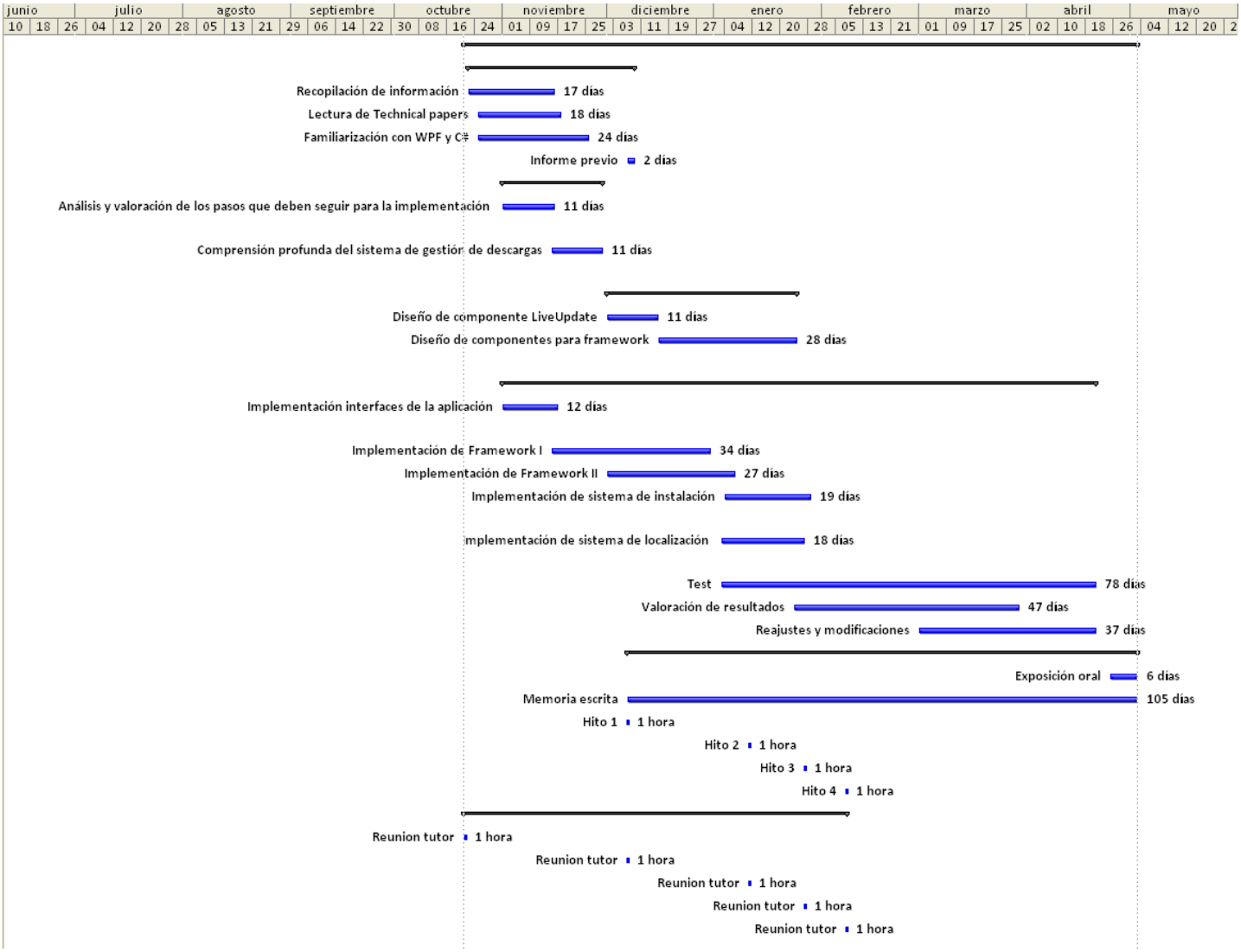

Figura 2.4: Diagrama de Gantt resultante del estudio de viabilidad.

Cuando lleguemos a la finalización del proyecto sería interesante poder hacer una comparativa entre este diagrama de Gantt y lo que realmente se ha seguido pudiendo así obtener el porcentaje en desviación que ha habido.

2.6 Evaluación de riesgos

La consecuencia más destacada de la evaluación de riesgos pueden ser la suspensión o cancelación del proyecto, ya sea por temas económicos, de recursos o cualquier aspecto que haga que el gerente o jefe de proyecto decida su suspensión.

El cambio de requerimientos puede ser un riesgo de cara al proyecto. Llevar un ritmo de trabajo, semanas de desarrollo para que finalmente se cambien los requerimientos de la aplicación supone un retraso del trabajo y puede afectar al presupuesto asignado al proyecto.

Otro de los riesgos que existen es utilizar WPF, no podemos asegurar al 100%, aun disponiendo de experiencia en otros proyectos, que no puedan surgir posibles problemas fruto de su utilización.

Al basarse principalmente en la descarga de actualizaciones vía internet la infraestructura del cliente también es un factor importante. Por eso, debemos minimizar los posibles problemas por una conexión a Internet lenta o con cortes.

2.7 Análisis de costes - beneficios

El mercado internacional de sistemas fotográficos para profesionales está altamente fragmentado. Además la información disponible sobre el mercado no es homogénea, ya que por ejemplo, en algunos mercados claves (Brasil), se han contado instalaciones de sistemas *Kiosk Gifts* con 1000 Kioscos en mercado siendo este un número muy elevado de kioscos para una sola cadena de establecimientos.

La política de comercialización del nuevo producto prevé aumentar el mercado entrando en nuevos sectores de negocio como pueden ser las copisterías, centros comerciales, hoteles vacacionales, además de mantener el sector de la fotografía profesional como principal sector comercial.

La vía de obtención de beneficios se produce de la siguiente manera:

- Adquisición de un equipo Kiosco y como mínimo una impresora. El número de impresoras dependerá de las necesidades del cliente en cuanto a formatos de impresión; 10x15, 15x23, 20x30 y hasta más de 30 formatos).
- El equipo se proporciona con una versión de software Kiosk Gifts completa, aunque para disfrutar de todas sus opciones existe un sistema de activación de licencias.
- En los consumibles de las impresoras, tanto a nivel de tinta como de papel (ya que este producto solo se pueden adquirir en la compañía o distribuidores oficiales).

Podemos extraer de las diferentes vías de obtención de beneficios que el software *Kiosk Gifts* es el pilar que mueve el resto de vías. El cliente que desee obtener un equipo para su establecimiento optará por un kiosco u otro principalmente por su funcionalidad.

Es necesario hacer una valoración PFC, pero a la vez también hacer referencia a la parte que abarca el proyecto global. De esta manera los presupuestos del proyecto se dividen de la siguiente manera:

- **Presupuesto del módulo LiveUpdate** (comporta el 5% de ***Kiosk Gifts)***

Activos fijos:

Descripción Equipo	Coste (€)
Software Microsoft: - 3 x Microsoft Windows XP. - 3 x Microsoft Visual Studio 2010. - Microsoft Windows Server 2008. - 3 x Microsoft expression Blend 4. - 3 x Microsoft Office 2010.	11.000€
PCs: - 2 x Dell Intel Core 2 duo 2.53 con 3 GB de RAM. - Sony Vaio Intel Core 2 duo 2.53 con 3 GB de RAM. - PC para hospedaje de web service y ficheros (empresa): Máquina virtual, la cual permite asociar tantos recursos como sean necesarios (RAM, CPU, etc.). - PC encargado de la redirección de IP fija a máquina virtual (IP por la que accedemos a las descargas). Datos técnicos no especificados por la empresa).	6.000€
Kioscos: - Kiosco Mitsubishi (para demo) con Intel Core 2 duo 2.40 con 1 GB de RAM.	2.000€
TOTAL	19.000€

Costes directos de personal:

Nombre	Rol	Coste/hora (€)	Horas dedicación	Coste Total (€)
Oscar Sanz	Jefe global del proyecto	60	20	1.200€
José Martín	Soporte técnico	50	30	1.500€
Alex Arroyo	Analista-Programador	40	200	8.000€
Departamento Q.A	Testadores	30	80	2.400€
TOTAL				13.100€

Material fungible y costes indirectos:

Concepto	Coste (€)
Costes Indirectos del personal	20.220€
TOTAL	20.220€

Resumen presupuestario LiveUpdate:

Definición	Coste (€)
Activos fijos	19.000€
Personal	13.100€
Material fungible y costes indirectos	20.220€
TOTAL COSTE PFC	52.320€

- **Presupuesto general del proyecto Kiosk Gifts:**

Los datos económicos que se muestren en este apartado son totalmente reales; esta información ha sido proporcionada por diferentes departamentos de la empresa.

Activos fijos:

Descripción Equipo	Coste (€)
Software Microsoft	150.000€
TOTAL	150.000€

Costes directos de personal:

Nombre	Rol	Coste/hora (€)	Horas dedicadas	Coste (€)
Xavier Cantan	Jefe de R&D	75,00	500	37.500€
Albert Gómez	Jefe de R&D-Photo	60,00	1.728	103.680€
Oscar Sanz	Jefe de proyecto	60,00	3.456	207.360€
Joan Sellarés	Analista	50,00	3.456	172.800€
José Martín	Suporte técnico	50,00	3.456	172.800€
Marcos Varela	Analista-Programador	40,00	3.456	138.240€
Albert Franzi	Desarrollador	40,00	3.456	138.240€
Joan Vila	Analista-Programador	40,00	864	34.560€
Alex Arroyo	Analista-Programador	40,00	1000	40.000€
Departamento Q.A (2 personas)	Calidad	30,00	1.728	51.840€
TOTAL				1.097.020€

Material fungible y costes indirectos: (PCS, software, etc.)

Concepto	Coste (€)
Costes indirectos de personal	231.100€
TOTAL	231.100€

Resumen presupuestario del desarrollo del Kiosk Gifts:

Definición	Coste (€)
Activos fijos	150.000€
Personal	1.097.020€
Material fungible y costes indirectos	231.100€
TOTAL COSTE PROYECTO KIOSK GIFTS	1.478.120€

Costes de Implantación para cliente (para configuración mínima de venta al público):

CLIENTS (PCS, Impresoras, periféricos)				
Grupo	**Producto**	**Unidades**	**Precio unitario (€)**	**Subtotal (€)**
Armario	Mueble de Kiosco para soporte de kiosco e impresoras Flexidesk.	1	550€	550€
Kioscos	Terminal fotográfico PT-7000E.	1	2400€	2400€
Impresoras	Impresora de sublimación (para formato base 10x15)	1	1177€	1177€
Impresoras	Impresora de tickets de caja para pedidos.	1	200€	200€
Consumibles	Consumible 10x15 (800 copias)	1	79€	79€

Importe	4206€
20 % de beneficio	1051€
IVA	18%
Importe con IVA	6.411€

A los costes de implantación se le suma un 20% (valor inventado, puesto que el valor real no ha sido aportado debido a la política de confidencialidad de Mitsubishi) que será la ganancia de Mitsubishi para cada una de les instalaciones que se haga del producto más las licencias que el cliente haya contratado:

Ingresos por implantación = 6.411,64 €
Beneficio (20%) por implantación: = 1.051,50 €

Licencies de uso (para activación de software complementario)				
Licencia activación Plus I- software gifts	Activación de software de calendarios y felicitaciones	1	400€	400€
Licencia activación Plus II- instant photo album	Activación de software de creación de libros fotográficos digitales.	1	400€	400€
Licencia activación Plus III- Easy Album	Activación de software de creación de álbumes fotográficos digitales	1	400€	400€

Importe	1200€
IVA	18%
Importe con IVA	1.416€

Configuración y puesta en marcha				
Rol	**Núm. Persones**	**Coste/hora (€)**	**Total horas**	**Coste (€)**
Técnico	1	50	2	100€
TOTAL				100€

Beneficio por instalación = Ingresos – Costes

= [Margen por implantación + Licencies de uso] – Costes Impl. = 1068,60 + 1.416,00 – 100 = **2.384,60€**

Previsión de ventas:

El número de instalaciones previstas es el siguiente:

Año	2010	2011	2012	2013	2014
España	0	50	75	75	100
Alemania	0	100	100	100	100
Holanda	0	25	50	50	50
UK	0	25	25	25	25
Francia	0	25	25	25	25
Italia	0	25	25	25	35
Resto de Europa	0	100	100	100	100
Asia	0	10	20	20	25
Latinoamérica	0	200	200	200	200
TOTAL	0	560	620	620	660

Haciendo balance de las instalaciones previstas, la tabla de ganancias por año queda de la siguiente manera:

Año	2010	2011	2012	2013	2014
Instalaciones	0	560	620	620	660
Ganancia instalación/ año	0	1.335.376€	1.478.452€	1.478.452€	1.573.836€
Costes presupuesto	739.060€	739.060€	0	0	0
Total ganancia por año.	-739.060€	596.316€	1.478.452€	1.478.452€	1.573.836€
TOTAL BENEFICIO: 4.387.996€					

Durante el primer año de desarrollo no se obtuvieron beneficios ya que se restó a cada uno de los dos primeros años los costes del desarrollo, quedando el primer año con cantidades negativas.

Ya en el año 2012 el producto quedará completamente amortizado. Finalmente el beneficio será de 4.387.996€

Se debe tener en cuenta que el beneficio no solo se obtiene de la venta directa de software, sino también por los equipos hardware y los consumibles de papel y tinta (en el caso de los consumibles solo tenemos en cuenta la primera venta, no el beneficio que generan durante toda la vida del producto).

Desde el departamento de Marketing se han realiza estudios de mercado entre nuestros clientes para determinar que beneficio aportaría la implementación de LiveUpdate llegando a la conclusión de que proporcionando una aplicación que mantenga siempre actualizado el software kiosk Gifts con parches y nuevas funcionalidades se realizarían un 15% más de ventas de equipos, con esto extraemos que si el beneficio final es de 4.387.996 € el beneficio de LiveUpdate (15 % sobre el total) será 658.199€ mientras que el coste del desarrollo asciende a 54.320 €.

2.8 Conclusión

Este proyecto está enmarcado dentro de otro, es decir, forma parte de un conjunto de módulos que con su integración formarán Kiosk Gifts. Sabiendo que Kiosk Gifts como producto final será viable, podemos tener ciertas garantías de que el módulo LiveUpdate también lo será.

Como consecuencia de la evaluación técnica y teniendo en cuenta los recursos de los cuales disponemos, el impacto económico que supone, costes-beneficios y la planificación temporal que se ha realizado, se puede afirmar que el proyecto es viable, y que su desarrollo puede continuar con garantías de que se finalice con las mínimas desviaciones y del modo en que se había planificado inicialmente.

3 ANÁLISIS

En este capítulo se presenta el análisis del problema que se desea resolver. Mostraremos un resumen del sistema actual de trabajo y una propuesta de mejora así como una serie de especificaciones funcionales y no funcionales que nos describirán que debe hacer el sistema y como debe hacerlo.

Dentro de este análisis también se estudiarán las opciones que nos ofrece el marco tecnológico para resolver los requerimientos y tener argumentos para valorar qué conjunto de herramientas se quieren utilizar para llevar a cabo el proyecto.

Finalmente se estudiarán las diferentes tecnologías de software para ver de qué manera organizamos la distribución de funcionalidades que ha de proporcionar la aplicación.

3.1 Sistema actual

El sistema actual de descargas ha quedado obsoleto en diversos sentidos, como por ejemplo visualmente. Por ello se desea la implementación de un nuevo sistema que disponga de una interfaz más atractiva y fácil de usar además de una funcionalidad más acorde con las necesidades actuales y futuras.

Actualmente el sistema permite descargar actualizaciones de software, pero se ve muy limitado por diversos factores. Los principales son:

1.- Dispone de una interfaz gráfica simple, por tanto no obtenemos toda la información que puede ser interesante para el usuario, no tenemos la visibilidad de un progreso ni un histórico de descargas.

2.- El tamaño de las actualizaciones aumenta con la aparición de nuevas versiones y actualmente no se implementa ningún gestor que controle su descarga, lo que indica que si se produce un problema durante la descarga se deberá reiniciar de nuevo desde cero.

3.- No implementa una funcionalidad que descargue automáticamente actualizaciones que solucionarían problemas críticos para el sistema, el sistema siempre requiere de la intervención manual para actualizarse.

4.- El actual sistema de actualizaciones esta implementado en lenguaje de programación Delphi mientras que el nuevo software (Kiosk Gifts) está enteramente desarrollado en WPF y lenguaje C#.

Dividiremos la casuística en módulos que iremos desgranando a lo largo de este apartado:

3.1.1 Interfaz del gestor de descargas.
3.1.2 Obtención de la información de las actualizaciones.
3.1.3 Gestión de las actualizaciones en descarga.
3.1.4 Visionado de las descargas finalizadas e instaladas.
3.1.5 Configuración del sistema.

3.1.1 Interfaz del gestor de descargas

La actual pantalla de actualización puede resultar algo confusa para los usuarios más noveles, sobre todo teniendo en cuenta que los operadores pueden tener pocos conocimientos de informática. Además, la aplicación no dispone del **look & feel** utilizado en Kiosk Gifts, ni soporte **multiidioma**, siendo el inglés el único idioma soportado. Finalmente, otra de las limitaciones es la falta de información de las nuevas actualizaciones para el kiosco.

Ahora mostraremos el actual interfaz de gestión de descargas (Fig. 3.1):

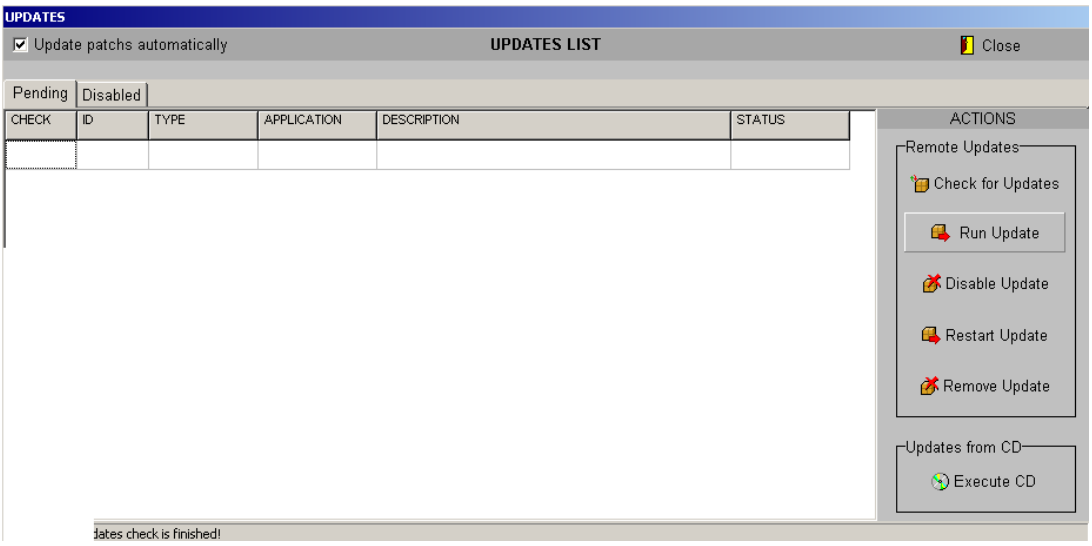

Figura 3.1: Sistema actual de descargas.

Para cubrir las necesidades definidas con anterioridad, el nuevo sistema deberá tener una interfaz simplificada, de uso sencillo y agradable. Además, dispondrá una estética acorde con el resto de

aplicativos de Kiosk Gifts donde se dispondrá de varios estilos seleccionables por el usuario en tiempo real. El sistema también dispondrá de un sistema de localización multiidioma fácilmente configurable que permitará al usuario modificar el idioma de la aplicación en tiempo real. Finalmente, en la pantalla principal de la aplicación se incluirá un control web que mostrará una web corporativa con información sobre nuevas actualizaciones y productos que vayan apareciendo.

3.1.2 Obtención de la información de las actualizaciones

Actualmente, el usuario final ve limitadas las opciones que presenta este aplicativo. Uno de los principales problemas es la confusión que crea en el usuario una interfaz que mezcla acciones de obtención de actualizaciones con otras de gestión de las descargas (*restart*, *remove*). El nterfaz no dispone de información al ejecutar sus opciones, no aportando ninguna información de progreso mientras realiza procesos lentos como puede ser la obtención de la información de las actualizaciones.

A continuación veremos el panel de opciones del actual sistema de actualizaciones (Fig. 3.2):

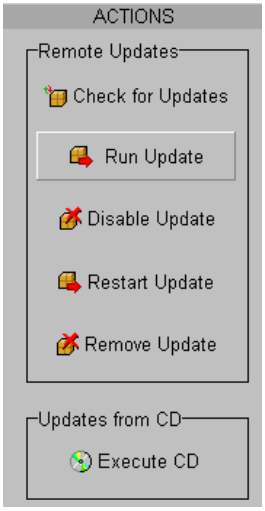

Figura 3.2: Panel de opciones del actual sistema de actualizaciones

Actualmente, no se realiza ninguna búsqueda automática diaria de actualizaciones, dejando al usuario final la responsabilidad de actualizar las aplicaciones con los parches más críticos. Además carece de soporte para lo que denominaremos actualizaciones de tipo *"Silent"* (actualizaciones transparentes para el usuario, no visibles en el interfaz de la aplicación) ni para las de tipo *"Autodownload",* que son actualizaciones que pasan a descargarse justo después de obtener su información.

El sistema actualmente tampoco permite realizar una multiselección de las actualizaciones para añadir todas ellas a descargar de un modo rápido.

La información que proporciona la aplicación respecto a la actualización es limitada esto da al usuario una sensación de desinformación generalizada que puede resultarle frustrante. A continuación la información de la actualización mostrada actualmente (Fig 3.3):

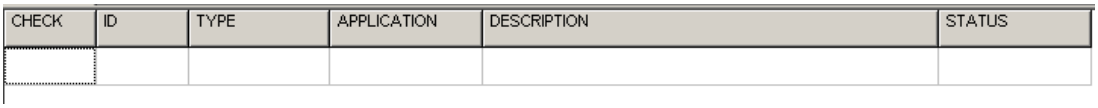

Figura 3.3: Información de la actualización mostrada actualmente.

Actualmente tampoco se dispone de una opción para "limpiar" la información de una actualización concreta.

Finalmente otro de los puntos que se deberían mejorar en el sistema es la obtención de descargas mediante soporte físico (actualmente DVD), por ser la actual funcionalidad muy limitada. El sistema podría mejorarse añadiendo nuevos soportes de obtención de la información como son las tarjetas de memoria o los USB.

Para cubrir estas necesidades se deberá implementar una opción (*"Actualizaciones"*) que únicamente contenga funcionalidad para la obtención de las actualizaciones y que sea independiente a la descarga.

Relacionado con el problema de interacción (*"feedback")* del actual sistema, se implementará un control que deberá mostrarse en las operaciones más costosas (en tiempo), como pueden ser la obtención de las actualizaciones "Vía Web" o el "Añadir a descarga".

Se implementará un sistema de gestión para la obtención de actualizaciones que se ejecute automáticamente a diario y que contemple el soporte de nuevos tipos de actualización.

Además deberemos permitir la multiselección de las actualizaciones y el limpiado temporal de las mismas.

Se deberá implementar un control que contenga información más útil que la actual para el usuario. Por ello, mantendremos parte de la información actual y le añadiremos nueva, como pueden ser las *"Release Notes"* o el tamaño de la descarga. Por temas de espacio en la interfaz de nuestra aplicación utilizaremos otro control para mostrar el contenido total de las *"Release Notes".*

Finalmente, extenderemos la funcionalidad actual de la obtención vía DVD a cualquier dispositivo de almacenamiento (DVD, lápiz de memoria y tarjeta de memoria) que detecte el explorar de archivos de Windows.

3.1.3 Gestión de las actualizaciones en descarga

Actualmente este módulo de la aplicación es el más limitado en cuanto a funcionalidad, debido a que no existe un gestor de descargas propiamente dicho. Las actualizaciones que se realizan son únicamente de un solo fichero. Esto provoca que si una descarga falla al 99% deberemos volver a descargar toda la actualización desde cero. Además, no existe en este momento una gestión de descargas de tipo *"Silent"*. Tampoco se permite en la actual versión la descarga de actualizaciones en paralelo descargándose estás siempre en orden de llegada (tipo FIFO).

El sistema actual no se permite la pausa o reanudación de una descarga, por ello no podemos primar una actualización respecto a otra.

El usuario debería disponer de más control sobre las actualizaciones pudiendo seleccionar, deseleccionar o limpiar (siempre de modo temporal) las descargas ya finalizadas en el sistema. Tampoco se dispone de información en el momento que se realiza la instalación del paquete descargado.

Finalmente, el sistema actual dispone de una gestión de la instalación de las actualizaciones muy limitado, en este, se instalan únicamente en el momento de la finalización o en el posterior reinicio del equipo.

Por ello, deberemos implementar en primer lugar un nuevo sistema de descarga por partes, desde el cual podamos descargar un fichero fraccionado en n partes y después volver a unirlo para su posterior instalación. El sistema debe permitir la descarga de actualizaciones en paralelo, donde el usuario pueda definir la proporción y teniendo en cuenta una serie de limitaciones, ya que la proporción de descargas estándar debe ser mayor que las de tipo *"Silent"*. Este nuevo sistema de gestión permitirá la implementación de las opciones "pausar" y "reanudar", de ese modo podremos detener y reanudar en todo momento una descarga en progreso.

También se implementará un sistema de reintentos para los casos en que falle la descarga de una parte, el número de intentos será configurable, tanto el número de reintentos por parte como el número máximo de ellos que se pueden realizar diariamente. Después de sobrepasar el número de intentos diarios el sistema dejará de reintentar descargar cualquier parte fallida.

Aumentaremos el control de las descargas por parte del usuario permitiéndole seleccionar/deseleccionar o limpiar las descargas. También se implementará un control que deberá mostrarse en las operaciones más costosas en tiempo, como puede ser la instalación de una descarga.

Finalmente se deberá implementar un sistema de gestión para la instalación de las descargas ya finalizadas. Se implementará una gestión integral de la instalación realizando tareas de *pausa* y *reanudación* de los servicios y procesos necesarios antes y después de la instalación. La información de que procesos o servicios se deben operar para una descarga estará en el paquete del instalador en forma de fichero xml.

3.1.4 Visionado de las descargas finalizadas e instaladas

Actualmente solo existe una diferenciación para las descargas finalizadas, el *"status"*, esto hace que no se cumpla una de las premisas que marcamos al inicio del desarrollo: que el interfaz sea lo más sencillo y claro posible.

Además, la aplicación no permite acceder al fichero físico descargado, ni eliminar únicamente la información de la descarga en caso de esta ser incorrecta.

Por ello, se debe implementar un nuevo sistema de visionado de descargas instaladas. Así que para solucionar los problemas anteriormente expuestos se procederá a separar en dos listas, una para las instalaciones que hayan finalizado correctamente y otra para las finalizadas incorrectamente. De este modo habrá un rápido reconocimiento visual:

- Las descargas instaladas correctamente con un color verde.
- Las instaladas incorrectamente con color rojo.

Además se procederá a añadir las opciones de "Acceso a fichero" para cada una de las descargas instaladas correctamente y las opciones "Acceso a fichero" y "Borrar Información" para cada una de las instaladas incorrectamente.

Finalmente, debido a la gran cantidad de descargas que pueden existir instaladas en el sistema, la información debe poder mostrarse de un modo sencillo aplicando, por ejemplo, una barra de "scroll" de tipo cinético.

3.1.5 Configuración del sistema

Actualmente solo existe un parámetro de configuración en el sistema actual de actualizaciones, "Update patchs automatically". Este queda obsoleto con la nueva implementación de gestión de descargas ya que esta información se definirá cuando creemos el paquete de la descarga. A continuación mostraremos el parámetro de configuración de la aplicación actual (Fig 3.4):

Figura 3.4: Configuración de la aplicación actual.

Debido a que la aplicación actual no tiene soporte multiidioma no existe ningún apartado donde podamos modificar el idioma de la aplicación. Al no tener soporte de estilos tampoco existe ningún parámetro de configuración del estilo de la aplicación. Tampoco encontramos ningún parámetro que especifique si es necesario iniciar las descargas pausadas después de iniciar el sistema.

Por ello, se implementará el soporte de configuración multiidioma en tiempo de ejecución y una nueva opción con la que el usuario podrá modificar la apariencia de la aplicación en tiempo de ejecución. Se podrá elegir en una primera versión entre dos estilos:

- Normal (estilo por defecto).
- Alto contraste (estilo opcional).

El otro punto a implementar será la opción con la cual el usuario podrá configurar si desea o no que después de reiniciar la aplicación se reanuden las descargas pausadas.

3.2 Requisitos funcionales

Para describir las funcionalidades que ha de incluir la aplicación se describirán los requerimientos de la obtención de la información de las actualizaciones, de la gestión de las actualizaciones/parches en descarga, del visionado de las descargas finalizadas (e instaladas) y la configuración del sistema.

3.2.1 Interfaz del gestor de descargas

El conjunto de funcionalidades que han de estar disponibles en el nuevo gestor del interfaz de descargas son:

• Creación de una interfaz atractiva y simplificada para cualquier perfil de usuario.
• Carga de Web corporativa con información para el cliente.

3.2.2 Obtención de la información de las actualizaciones

Los requerimientos para obtención de la información de las actualizaciones son:

• El usuario dispondrá de la información de las actualizaciones y nuevos productos, cuando la información ya haya sido obtenida previamente.
• Diariamente se realizará una búsqueda automática de nuevas actualizaciones en el servidor.
• El usuario podrá obtener la información de las actualizaciones (vía web) en el momento que desee mediante una opción en pantalla.
• El usuario podrá cargar la información de las actualizaciones (almacenadas en un USB o HDD) mediante una opción en pantalla.
• El usuario siempre podrá seleccionar que actualizaciones de la lista desea que se descarguen mediante las opciones en pantalla.
• En caso de que una actualización sea crítica deberá empezar su descarga inmediatamente, sin esperar la autorización del usuario.
• El usuario tendrá la opción de seleccionar/ deseleccionar todas las actualizaciones.
• El usuario podrá limpiar (siempre de modo temporal) la información de las actualizaciones.

Estos requerimientos han sido diseñados para cumplir los siguientes casos de uso (Fig. 3.5 y 3.6):

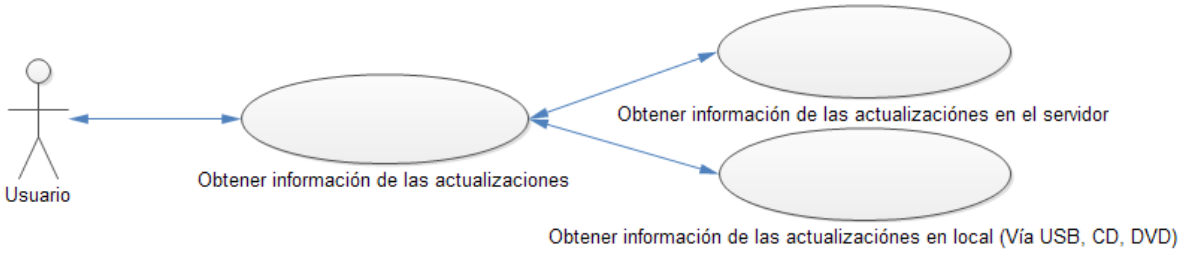

Figura 3.5: Caso de uso para la obtención de la información de las actualizaciones.

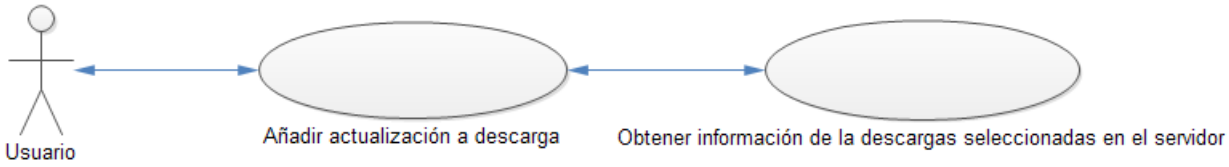

Figura 3.6: Caso de uso para la acción de añadir actualizaciones a la lista de descargas.

3.2.3 Gestión de las actualizaciones en descarga

El conjunto de funcionalidades que han de estar disponibles para la gestión de las actualizaciones es:

• Se implementará un nuevo sistema de descarga por partes, desde el cual se podrá descargar un fichero fraccionado en n partes y posteriormente unirlo en el cliente para su instalación.
• El sistema debe permitir descargas en paralelo. El número de partes que deseamos descargar a la vez debe ser configurable desde un fichero (siendo 3 por defecto).
• El sistema debe permitir descargas de tipo “Silent”.
• La proporción de las descargas “Silent” frente a las estándar debe ser configurable por el usuario. Se debe tener en cuenta la limitación de siempre tener un valor superior en descargas estándar.
• El sistema de gestión debe permitir la pausa, el resumen y la cancelación de las descargas exceptuando las ya finalizadas.
• Se deberá implementar un sistema de reintentos, donde deberá ser configurable el número de intentos por parte y diarios que puede realizar el sistema.
• El usuario podrá seleccionar/deseleccionar o limpiar cualquier descargar de la lista.
• Deberá mostrarse un control durante las operaciones más costosas.
• Sistema de gestión para la instalación de las descargas ya finalizadas. Se deberán ejecutar las actualizaciones de tipo “Autoinstall” o “Silent” de modo automático, sin necesidad de conformidad por parte del usuario.
• Se implementará la gestión integral de la instalación, realizando tareas de pausa y reanudación de los servicios y procesos necesarios antes y después de la instalación.
• La información de la instalación estará en el paquete del instalador en forma de fichero xml.

Los requerimientos de este apartado se ilustran el siguiente caso de uso (Fig. 3.7):

Figura 3.7: Caso de uso para la gestión de la descarga.

3.2.4 Visionado de las descargas finalizadas e instaladas

Estos son el conjunto de funcionalidades que han de estar disponibles para la visualización de las descargas finalizadas (e instaladas):

• Se implementará un sistema de visualización dividido en dos listas: la primer para las instaladas correctamente y la segunda para las instaladas incorrectamente.
• El sistema implementará un sistema de colores para un rápido reconocimiento del tipo de instalación: Verde para correctos y rojo para incorrectos.
• Se añadirán opciones de “Acceso a fichero” para cada una de las descargas instaladas correctamente.
• Se añadirán opciones de “Acceso a fichero” y “Borrar Información” para cada una de las descargas instaladas incorrectamente.
• Debido a la gran cantidad de descargas que pueden existir instaladas en el sistema, se implementará un control deslizante (a modo de scroll cinético).

Los requerimientos de este apartado se ilustran en el siguiente caso de uso (Fig. 3.8):

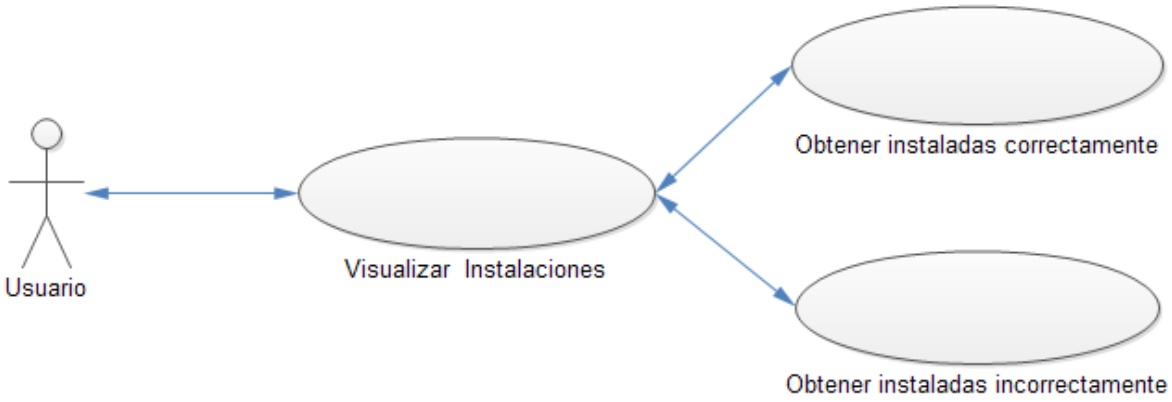

Figura 3.8: Caso de uso para la visualización de las descargas instaladas.

3.2.5 Configuración del sistema

Las funcionalidades que permiten la gestión de la configuración del sistema son:

• Soporte de configuración multiidioma en tiempo de ejecución.
• El usuario podrá configurar si desea que después de reiniciar la aplicación se reanuden las descargas pausadas.
• El usuario podrá modificar la apariencia de la aplicación en tiempo de ejecución.
• El fichero de configuración permanecerá en todo momento encriptado en el sistema. Debido principalmente a que contiene rutas importantes para la aplicación.

Los requerimientos de este apartado se ilustran en el siguiente caso de uso (Fig. 3.9):

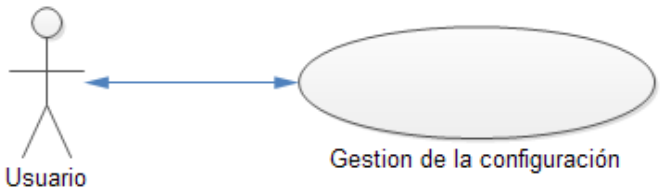

Figura 3.9: Caso de uso para la gestión de la configuración.

3.3 Requisitos no funcionales

Existe un conjunto de especificaciones relacionadas con el entorno en el que debe funcionar la aplicación y que son necesarias en el cumplimiento de nuestros objetivos:

- Permitir acceso concurrente a la información de las descargas.
- Hacer uso de interfaces amigables y para usuarios que no necesariamente deben ser expertos en informática.
- Garantizar una comunicación estable entre el cliente y el servidor cuando se realicen descargas de actualizaciones.
- Proporcionar una velocidad de descarga suficiente (en el lado de servidor) para que un usuario disponga de una actualización en un tiempo razonable.
- Gran potencia gráfica en la aplicación y tecnología orientada a objetos.

3.4 Marco tecnológico

Para determinar cómo se debe llevar a cabo la implementación se realizará un análisis del marco tecnológico. Al analizar las diferentes herramientas de las que disponemos para llevar a cabo la aplicación se han tenido en cuenta que se plantea utilizar una arquitectura de tres capas: lógica, pantallas y datos. Se deben permitir accesos concurrentes de diferentes usuarios a una aplicación servidor que sirva los datos: Información y partes de las actualizaciones.

Resulta imprescindible la posibilidad de disponer de programación orientada a objetos. Este tipo de programación es muy útil cuando se trata de desarrollar entidades bien definidas formadas por una serie de atributos y métodos. Otros de los puntos del marco tecnológico serán:

3.4.1 Lenguaje de programación.
3.4.2 Gestión de control de código fuente.

3.4.1 Lenguaje de programación

Al proporcionar una amplia gama de funciones en una sola tecnología, WPF simplifica de forma significativa la creación de interfaces de usuario modernas. Gracias a la unificación en una misma base de todas las tecnologías necesarias para crear interfaces de usuario, WPF puede simplificar enormemente la labor de quienes crean las interfaces. Sólo tendrán que familiarizarse con un único entorno, por lo que WPF puede reducir el coste asociado a la creación y el mantenimiento de aplicaciones. Además, al facilitar la generación de interfaces que incorporan gráficos y vídeo, entre otros elementos, WPF puede mejorar la calidad y el valor comercial de la interacción de los usuarios con las aplicaciones de Windows.

No obstante, la creación de interfaces de usuario modernas va más allá de la unificación de tecnologías diversas. También consiste en aprovechar las ventajas que ofrecen las tarjetas gráficas modernas. De este modo, WPF puede transferir la mayor carga de trabajo posible a cualquier unidad de procesamiento de gráficos (GPU) disponible en el sistema. Una interfaz moderna tampoco debe verse limitada por las deficiencias de los gráficos de mapa de bits. Por esta razón, WPF usa únicamente gráficos vectoriales, lo que permite que las imágenes se ajusten automáticamente al tamaño y a la resolución de la pantalla en la que se muestran. En lugar de crear gráficos diferentes para la presentación en monitores pequeños y en pantallas grandes, el desarrollador puede dejar que WPF se ocupe de adaptarlos.

A continuación, un diagrama que muestra la funcionalidad de la plataforma WPF (Fig. 3.10).

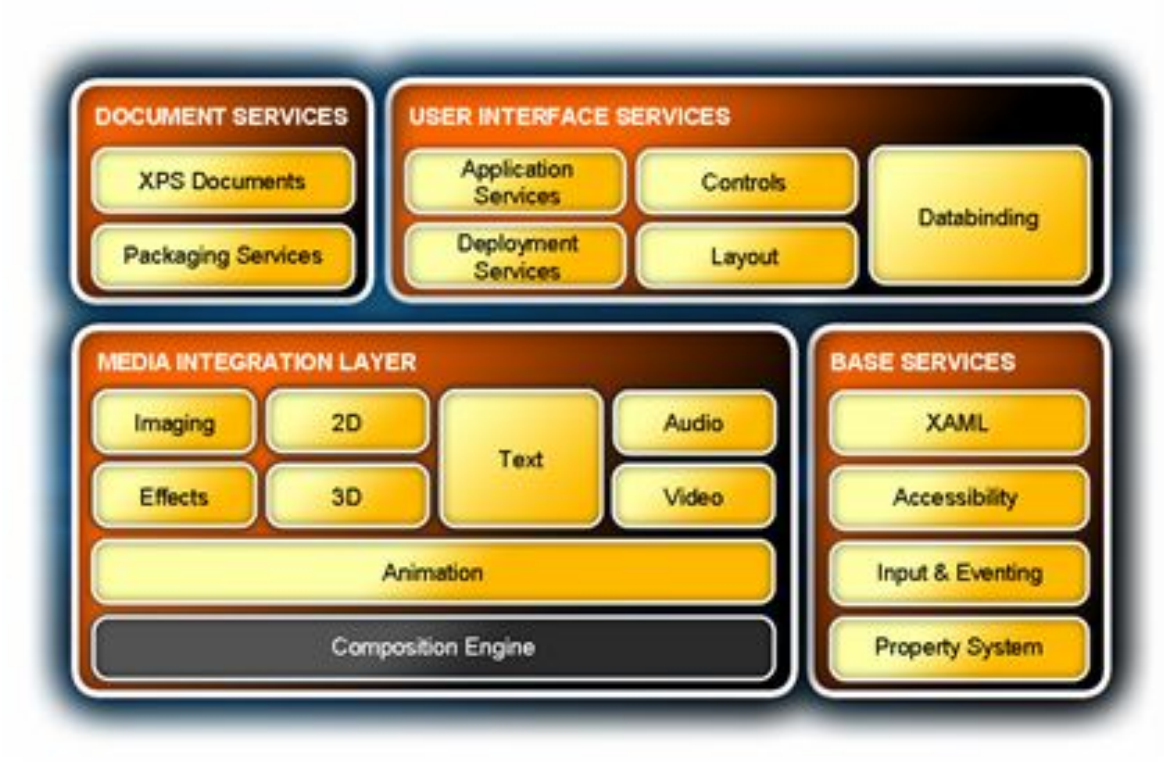

Figura 3.10: Bloques que forman WPF.

Posibilidad para desarrolladores y diseñadores de trabajar conjuntamente en la misma aplicación en paralelo.

WPF incluye el lenguaje de marcado de aplicaciones extensible (XAML). El lenguaje XAML define elementos XML, como Button, TextBox, Label, entre muchos otros, para especificar exactamente la apariencia de las interfaces de usuario. Cada elemento XAML corresponde a una clase de WPF. A su vez, cada atributo de dicho elemento cuenta con una propiedad o evento correspondiente en la clase. XAML ofrece un método basado en herramientas muy sencillo para describir interfaces de usuario y, de este modo, permite una mejor colaboración entre desarrolladores y diseñadores.

WPF ofrece la posibilidad de utilizar de las mismas tecnologías tanto para interfaces nativas de Windows como para interfaces de explorador Web. Así, un desarrollador puede crear una aplicación XAML del explorador (XBAP) con WPF, que se ejecuta en Internet Explorer. De hecho, es posible usar el mismo código para crear una aplicación de WPF independiente y una XBAP.

La descarga de XBAP se lleva a cabo a petición desde un servidor Web, por lo que los requisitos de seguridad asociados son más estrictos que en el caso de aplicaciones de Windows independientes. Por consiguiente, las XBAP se ejecutan en un recinto de seguridad proporcionado por la seguridad de acceso a código de .NET Framework. Por ejemplo, una XBAP implementada desde la zona de Internet no puede crear ventanas independientes o mostrar cuadros de diálogo definidos por la aplicación.

En cuanto a C# es el lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo.

3.4.2 Gestión de control de código fuente

Todas las modificaciones han de estar documentadas así que utilizaremos la herramienta de control de código fuente y archivos *Subversion*, que permite crear diferentes versiones de un archivo y documentar de modo breve la versión que se sube al servidor de manera que la próxima persona que lo utilice sepa quien ha sido el último en tocar un archivo en concreto y que cambios se han realizado. Esta aplicación viene a substituir a la anterior aplicación utilizada para la gestión del código fuente en nuestros desarrollos, *CVS*.

Las ventajas de *subversion* son:

- Se sigue la historia de los archivos y directorios a través de copias y renombrados.
- La creación de ramas y etiquetas es una operación más eficiente. Tiene coste de complejidad constante (O(1)) y no lineal (O(n)) como en CVS.
- Se envían sólo las diferencias en ambas direcciones, en CVS siempre se envían al servidor archivos completos.
- Maneja eficientemente archivos binarios a diferencia de CVS que los trata internamente como si fueran de texto.
- Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

La Fig. 3.11 muestra la estructura creada por el gestor de código fuente.

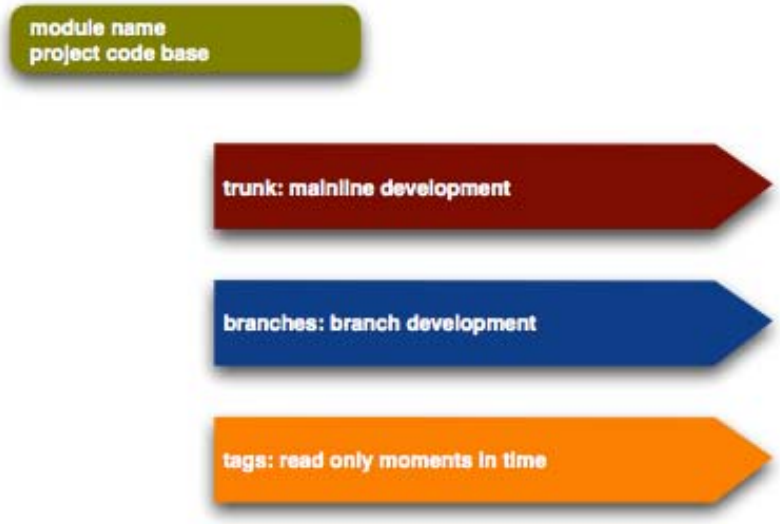

Figura 3.11: Árbol del proyecto SVN.

Las carencias son:

- El manejo de cambio de nombres de archivos no es completo, lo maneja como la suma de una operación de copia y una de borrado.
- No resuelve el problema de aplicar repetidamente parches entre ramas, no facilita llevar la cuenta de qué cambios se han realizado. Esto se resuelve siendo cuidadoso con los mensajes de commit.

3.5 Arquitectura de la aplicación

Para estructurar la arquitectura de la aplicación es más adecuado que se base en un modelo de tres niveles. Este tipo de estructura se define para la separación de las funciones de la aplicación en una capa para la presentación de la información (LiveUpdate), una para la lógica y funciones (Framework) y otra para los datos.

La capa de presentación incluye un conjunto de funciones que tienen que ver con la interacción entre el usuario y la aplicación. De esta manera se separa el diseño de la lógica permitiendo que otros aplicativos dispongan también de esa lógica.

La capa de lógica incluye la funcionalidad (Framework) que se recoge en parte de los requerimientos.

La capa de datos proporciona la información necesaria para el aplicativo, en este caso la información de las actualizaciones y sus partes.

A continuación mostraremos la arquitectura básica de la aplicación (Fig. 3.12):

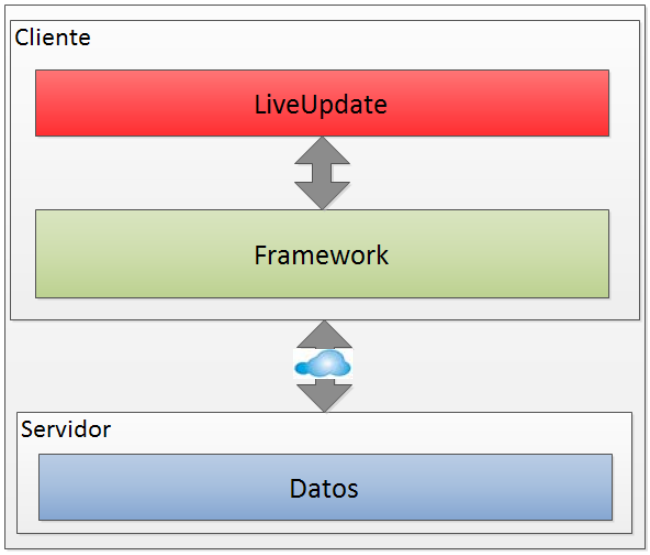

Figura 3.12: Arquitectura básica de la aplicación.

3.6 Conclusiones

Actualmente se dispone de un sistema de gestión de descargas anticuado debido principalmente a su estética, a la falta de información de las actualizaciones que proporciona el aplicativo y a la inexistente implementación de un gestor de descargas que permita pausar, resumir o cancelar una actualización en descarga, tampoco permite descargar automáticamente parches o aplicaciones para la solución de problemas críticos ni tiene un sistema integral de gestión de la instalación de la descarga.

Eso se puede mejorar con la implementación de una interfaz gráfica más moderna y actual, con el desarrollo de un gestor de descargas que cubra todas las necesidades actuales como puede ser la posibilidad de continuar una descarga sin finalizar después del apagado del equipo, además de un sistema de control integral de la instalación de una actualización ya descargada permitiendo así detener/arrancar procesos del sistema para la correcta instalación de nuestras aplicaciones. Todo esto utilizando un lenguaje orientado a objetos (C#) que permitirá crear módulos reutilizables por cualquier otro sistema que se desee desarrollar.

En el siguiente capítulo descubriremos los detalles de nuestra propuesta de diseño para el sistema LiveUpdate.

4 DISEÑO

En este capítulo desarrollaremos el diseño para nuestro aplicativo. Especificaremos cada uno de los componentes que formarán parte de Livepdate.

4.1 Introducción

En el capítulo de diseño definiremos los componentes necesarios para crear nuestro aplicativo y que este cumpla con el análisis descrito en el capítulo anterior. Los componentes serán:

- ***LiveUpdate,*** que contendrá los interfaces (como son los controles y recursos), la lógica con la gestión de pantallas y todos los componentes necesarios para la comunicación con el Framework.
- ***Framework***, que contendrá todos los componentes con posibilidades de ser reutilizados, como pueden ser el componente de comunicación online, controles genéricos o el sistema de localización. El componente Framework ha sido pensado para su utilización no solo por nuestra aplicación sino también por cualquier otra que en un futuro requiera de algún componente del mismo.

Además, realizaremos una revisión de la planificación actual del proyecto y especificaremos todas las posibles desviaciones que hayan podido aparecer a consecuencia del diseño.

4.2 LiveUpdate

En este apartado veremos los componentes de LiveUpdate y cómo estos interactuarán con el *Framework*. A continuación veremos los componentes de LiveUpdate*:*

- ***MainWindow,*** formulario principal de la aplicación. Albergará todas las *Views* (vistas).
- ***ViewModel,*** contiene toda la lógica de la aplicación para cada una de las funcionalidades (actualizar, descargar, descargas instaladas en el sistema o configuración). *MainWindowViewModel* es la clase encargada de gestionar que *ViewModel* se debe carga en cada momento dependiendo de qué opción pulse el usuario (para ello utilizará *Core*).
- ***View,*** representa todas las ventanas de la aplicación (Actualizar, Descargar, Instalables, Configuracion y Acerca de LiveUpdate). Las Views se relacionan con su lógica (*ViewModel*) mediante el módulo *Resources*, donde se indica que representación gráfica tiene cada lógica (ejemplo: la lógica *NewUpdatesViewModel* necesita NewUpdatesView para ser representada).
- ***UserControls,*** controles definidos para ser utilizados por las *Views*. Incluye entre otros, los controles que muestran la información de actualización, descarga o actualizaciones ya instaladas.
- ***Core,*** contiene la lógica que nos ayuda a gestionar la carga de los *ViewModel* (en *MainWindowViewModel*). También dispone de parte de la lógica del mediator, definiendo los mensajes entre los *ViewModels*.
- ***Resources,*** este módulo surte a la aplicación de los estilos e imágenes necesarios para su funcionamiento.
- ***Converters***, contiene la lógica de componentes que se encargan de transformar datos en la View en tiempo de ejecución (ejemplo: *transformDateToLocalization* se encarga de, en tiempo de ejecución, modificar el formato de las fechas a su correspondiente cultura.

En la siguiente figura 4.1 mostraremos las relaciones entre componentes (LiveUpdate y Framework):

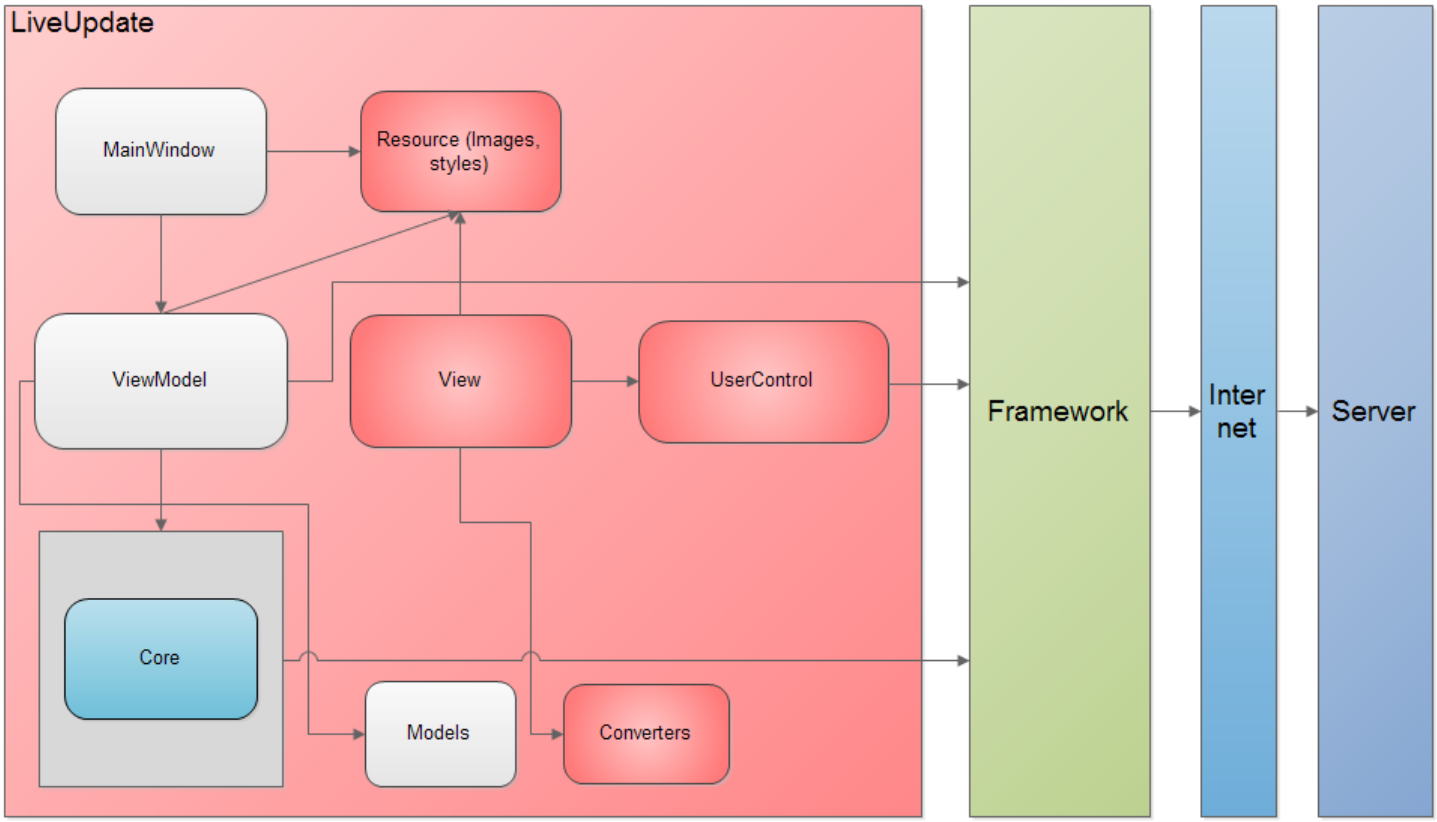

Figura 4.1: Diagrama donde se ven todas las relaciones entre los componentes de LiveUpdate.

Para mostrar la relación entre nuestro diseño y la funcionalidad de *LiveUpdate* dividiremos el aplicativo en sus principales módulos lógicos:

4.2.1 Actualizaciones.
4.2.2 Descargas.
4.2.3 Instalables.
4.2.4 Configuración.

4.2.1 Actualizaciones

Este es el módulo desde donde podemos obtener la información de las actualizaciones de las que dispone el sistema, ya sea obteniendo la información vía internet o a través de un dispositivo físico (CD, DVD o USB). También nos permite una vez obtenida la información de las actualizaciones decidir que actualizaciones deseamos descargar.

A continuación mostraremos un diagrama (Fig. 4.2) con los componentes que se utilizan en este módulo.

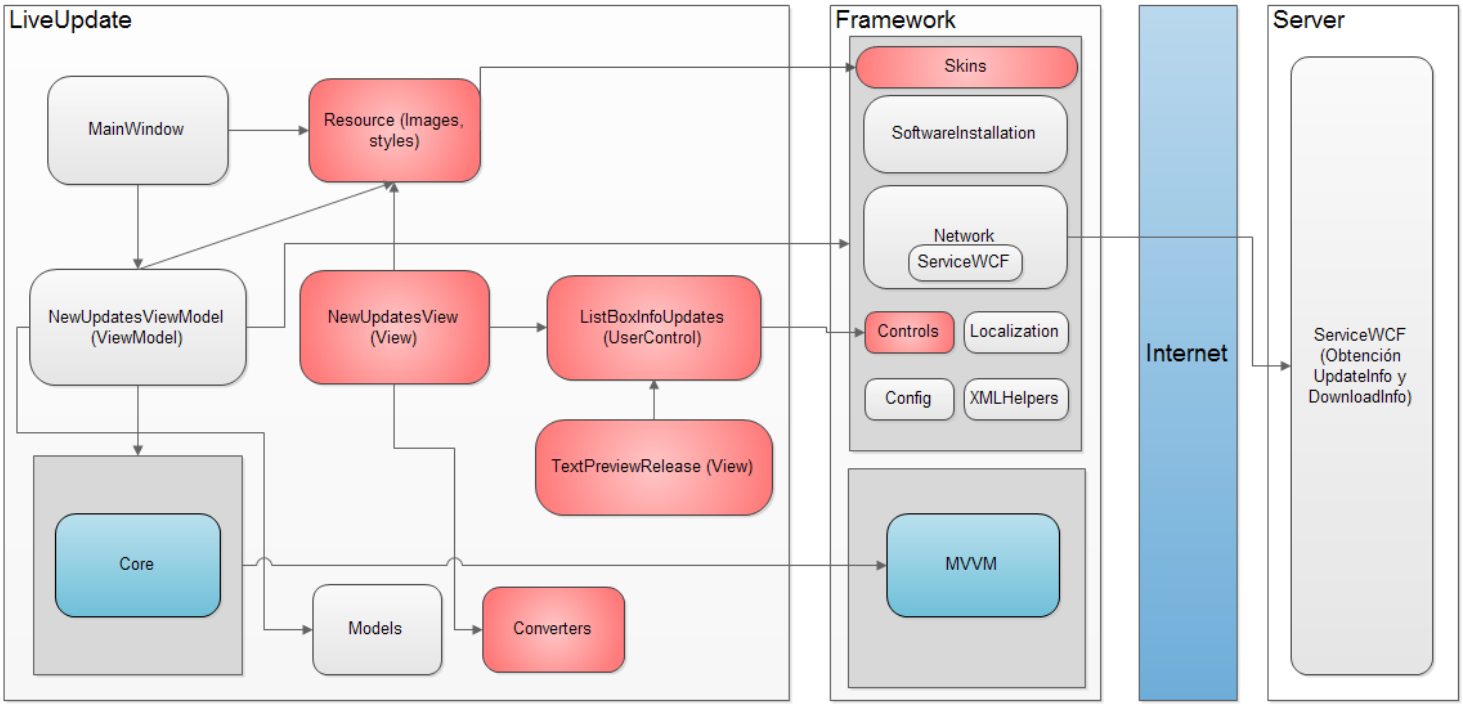

Figura 4.2: Diagrama con los módulos que intervienen en la obtención de las actualizaciones.

4.2.2 Descargas

Una vez seleccionadas que actualizaciones deseamos descargar, este módulo será el encargado de gestionar la descarga. Esto incluirá el soporte de la descarga por partes, pausa, reanudación y cancelación de la descarga. Internamente se gestionarán todos los tipos de descarga que existen:

- *Estándar*, descarga visible por el usuario y cuya instalación debe ser validada (mediante un botón de instalación).
- *AutoInstall*, una vez finalizada la descarga se procederá a su instalación.
- *Silent*, descarga no visible por el usuario (normalmente ligado a la auto instalación).

A continuación mostraremos un diagrama (Fig. 4.3) con los componentes que se utilizan en este módulo.

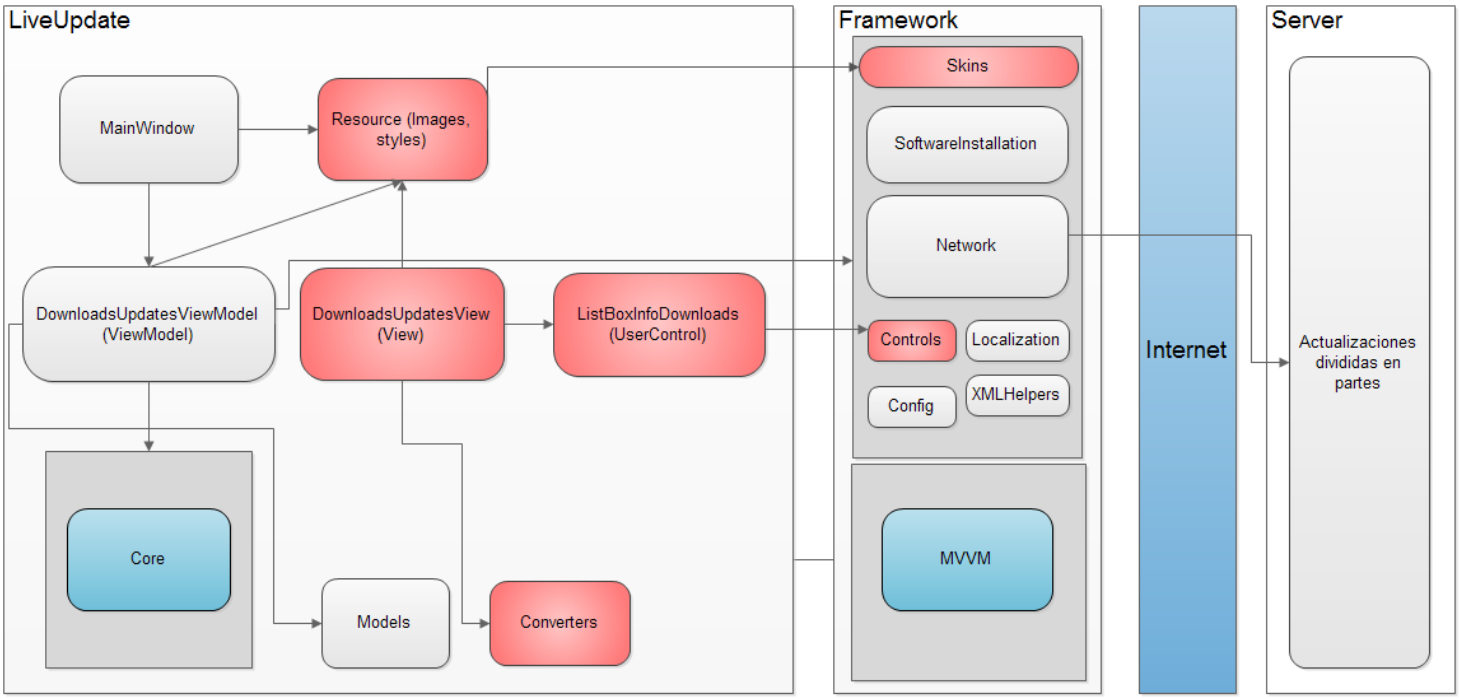

Figura 4.3: Diagrama con los módulos que intervienen en la descarga de las actualizaciones.

4.2.3 Instalables

En este módulo se mostrará la información tanto de las descargas instaladas correctamente en el sistema como en las que ha fallado la instalación. Además, se dispondrá de opciones para acceder a los ficheros descargados.

A continuación mostraremos un diagrama (Fig. 4.4) con los componentes que se utilizan en este módulo.

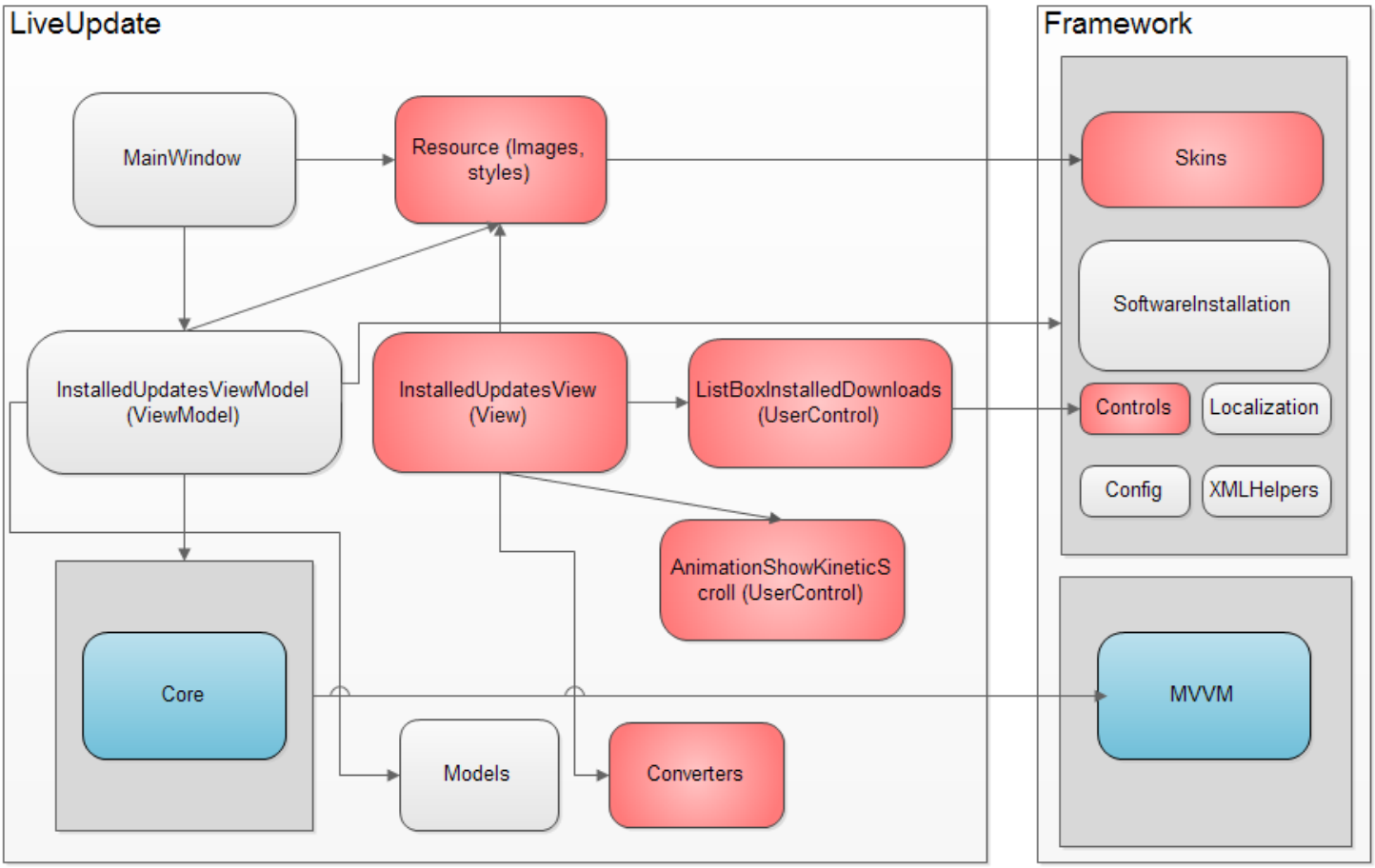

Figura 4.4: Diagrama con los módulos que intervienen en la visualización de la instalaciones.

4.2.4 Configuración

En este módulo se mostrarán diferentes apartados configurables desde nuestra aplicación: idioma de la aplicación, la activación/desactivación del sistema de descarga automática para descargas en estado de pausa al reiniciar el sistema y finalmente el estilo.

A continuación mostraremos un diagrama (Fig. 4.5) con los componentes que se utilizan en este módulo.

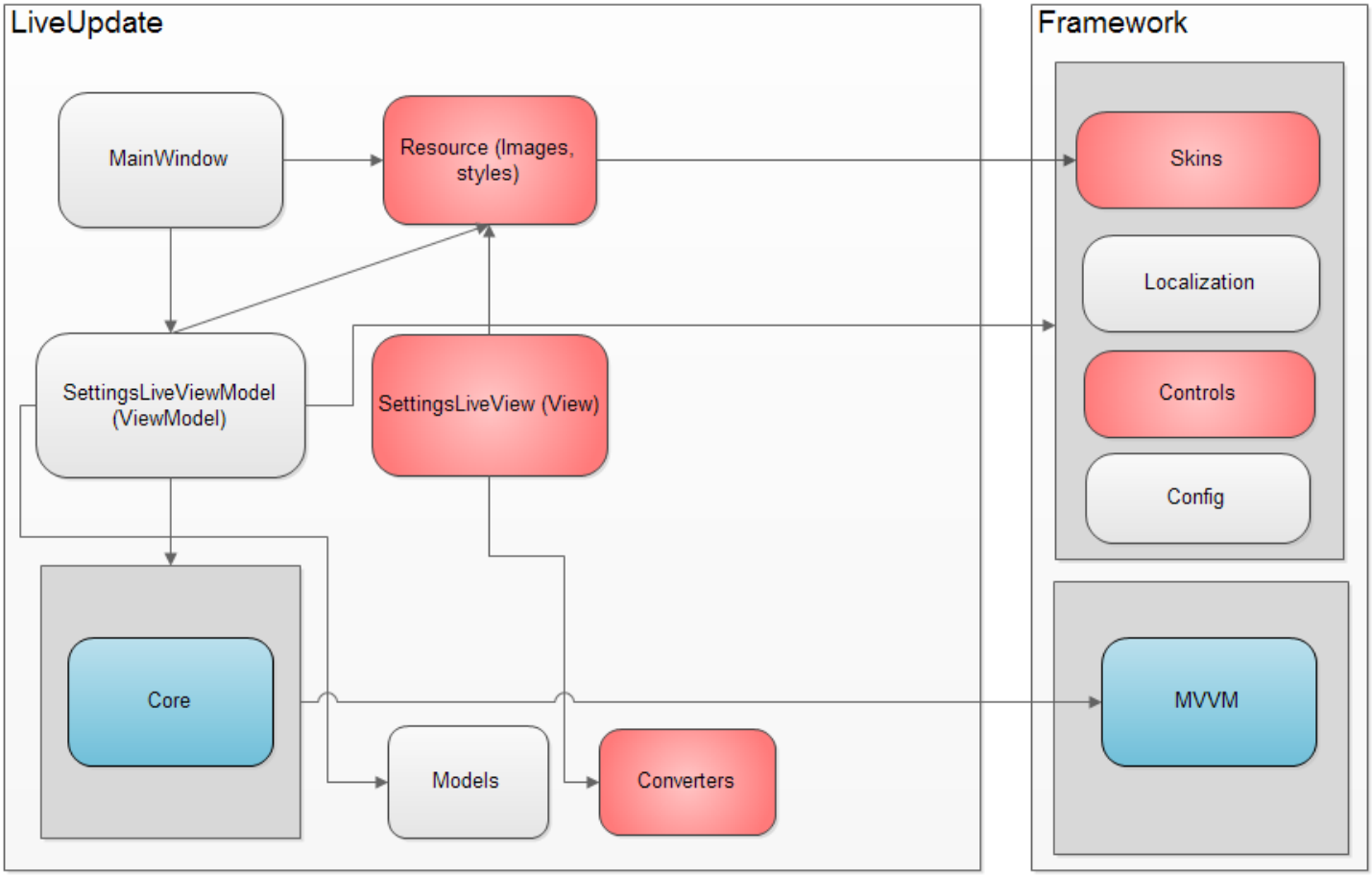

Figura 4.5: Diagrama con los módulos que intervienen en la configuración de la aplicación.

4.3 Framework

A la hora de diseñar 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 separar la vista del modelo de datos.

Se han diseñado los siguientes componentes:

- ***MVVM*** que aplica el patrón de diseño Model View ViewModel, sirve para separar la interfaz de usuario del modelo de datos. Este componente define cómo se deben declarar las ventanas de la aplicación. Requiere de ICommand (MSDN, 2010).
- ***Mediator*** implementado siguiendo el patrón de diseño con el mismo nombre (Erich Gamma, 1994) (Bishop, 2007), permite la comunicación entre componentes u objetos.
- ***Network*** es el componente que aportará toda la funcionalidad de gestión de las descargas. El componente implementará en la aplicación la capacidad de descargar diferentes partes de una actualización así como toda la gestión de los reintentos y la funcionalidad de "pausar" y "resumir" las descargas. También contendrá la clase que permitirá la deserialización del fichero xml que contendrá la información de la descarga (una representación virtual del fichero físico). Además contendrá una referencia al componente WCF y gestionará todas sus llamas.
- ***ServiceWCF*** es el componente que nos proporcionará la referencia a la implementación *ServiceWCF* del servidor. El componente permitirá obtener la información de la actualización y la descarga.
- ***Controls*** es el componente que aportará los controles para la representación de la pantalla de *"Release Notes"* y el reloj (a modo de *"feedback"*) utilizado en procesos que requieren de un tiempo mayor que la media.
- ***Sys*** es el componente que aportará toda la funcionalidad de gestión de servicios y procesos.
- ***SoftwareInstallation*** es el componente que aportará toda la funcionalidad de gestión de instalación de descargas, además de gestionar en la pre y post instalación las operaciones (arranque y parada) que se deben realizar sobre los servicios y procesos definidos en el fichero XML con la información de la instalación.
- ***Localization*** es el componente que se encargará de dar el soporte de multiidioma a toda la aplicación.
- ***Skins*** es el componente que se utilizará para establecer el estilo base de toda la aplicación del cual obtendremos algunas de las definiciones gráficas de nuestros componentes.
- ***XMLHelpers*** es el componente que se encargará de serializar/deserializar los XML que utilizamos en la aplicación
- ***InOut*** es el componente que se encarga de la partición/unión del fichero de actualización para su descarga y posterior unión. Además contendrá todo el sistema de lectura/escritura de ficheros en disco de nuestro aplicativo.

- ***Config*** es el componente que se encarga de gestionar el fichero de configuración donde se guardan todas las configuraciones de la aplicación.
- ***Cryptography*** es el modulo encargado de encriptar la información del fichero de configuración del sistema, así como de generar un CRC para cada una de las partes de la descarga para validar su integridad. Este sistema no se utilizará en ningún caso para validar si la parte es o no lícita o como sistema de seguridad, únicamente será un sistema de validación de integridad.

4.3.1 Model View ViewModel

Para conseguir la separación del modelo de datos con la vista, existe un patrón recomendado por Microsoft llamado Model View ViewModel. Por tanto, este componente 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 diseñador que busca una experiencia de usuario más exigente que los diseñadores tradicionales (Back end[1] o Business Logic). Como es en el caso de las plataformas de desarrollo de interfaces 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 contenidos en las mismas ventanas. 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.

La principal diferencia con el patrón de diseño MVC (Model View Controller) es que las entradas en MVVM se producen a través de la vista mientras que en MVC estas las gestiona el controlador. Además con el modelo MVC la capas de Model y Vista se conocen entre ellas debido a que la relación entre el modelo y el controlador es unidireccional, el controlador conoce el modelo pero no al revés.

[1] 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).

A continuación podremos ver un diagrama del MVVM y su comparación con el patrón MVC (Fig. 4.6).

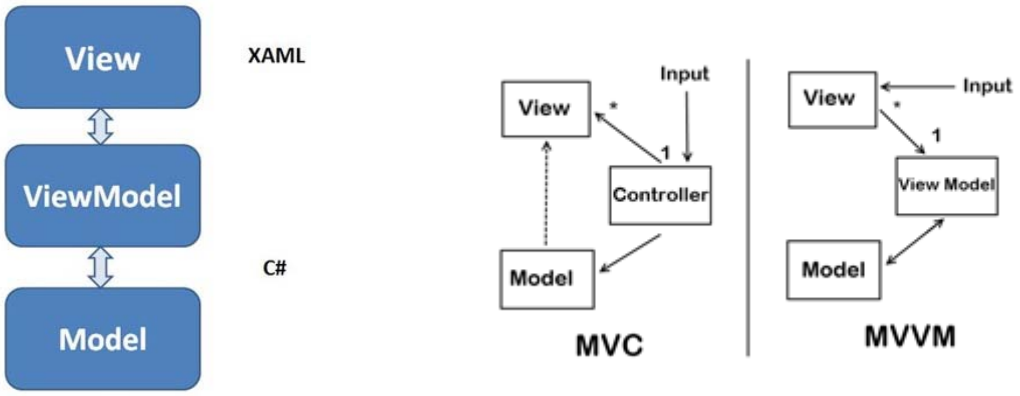

Figura 4.6: MVVM en un diagrama y su comparación con MVC.

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.

De manera más técnica, en MVVM:

- 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. Un comando es cualquier objeto que implementa la interfaz ICommand, la cual define tres miembros:
    - Execute: El método que ejecuta la lógica del comando.
    - CanExecute: Método que devuelve true, si el comando está habilitado, false si esta deshabilitado.
    - CanExecuteChanged: Evento que es disparado si el valor de CanExecute cambia.
- 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 en el proyecto.

4.3.2 Mediator

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

A continuación mostraremos un diagrama (Fig. 4.7) de cómo funcionará el Mediator en la aplicación.

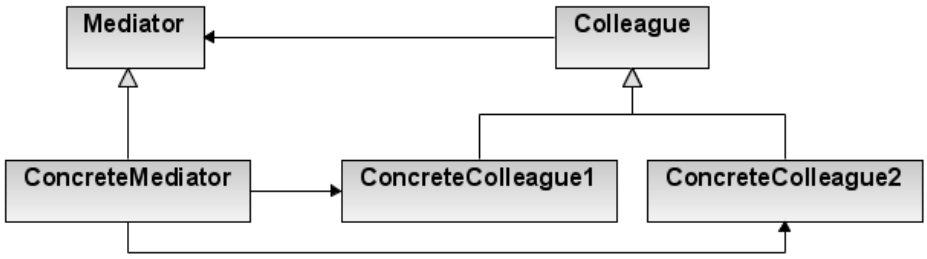

Figura 4.7: Patrón de diseño Mediator

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. Este componente, llamado WeakAction, utiliza WeakReference(Mayo, 2001) (MSDN, 2010) y se ha diseñado 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[2].

[2] Delegate: Tipo de clase de C# que permite ejecutar acciones (MSDN, 2010).

La lógica para añadir y obtener acciones está representada en los diagramas de estados representados en las figura 4.8.

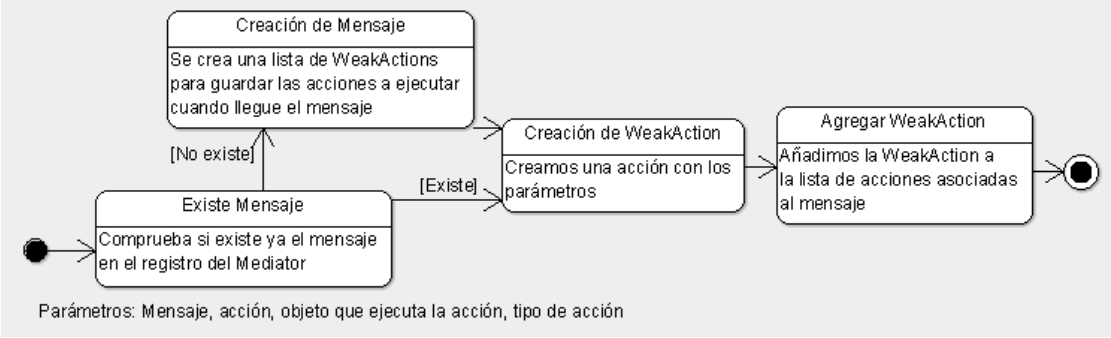

Figura 4.8: Diagrama de estados para añadir una acción.

Como se puede apreciar en la figura 4.8, cuando se añade una acción al Mediator, 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.

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.

A continuación mostraremos un diagrama de estados para obtener las acciones (Fig. 4.9).

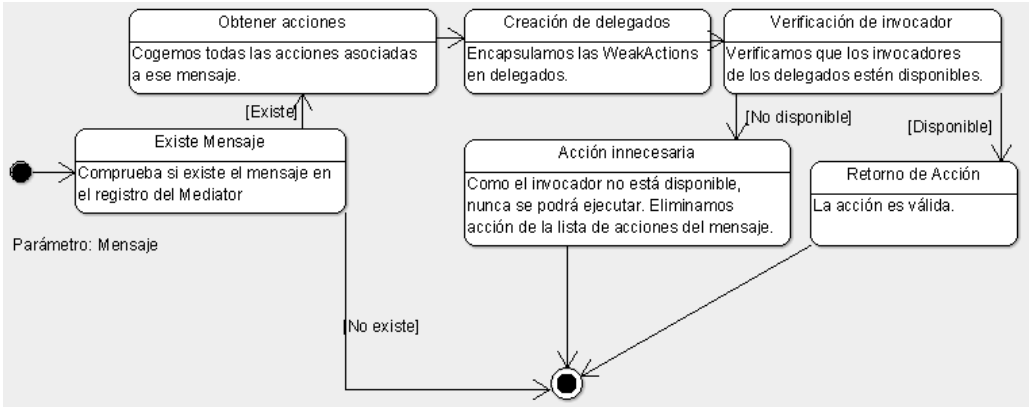

Figura 4.9: Diagrama de estados para obtener acciones

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.

4.3.3 Network

El módulo Network será capaz de ofrecernos la comunicación con el módulo alojado en el servidor (*ServiceWCF*), y así poder obtener la información de las actualizaciones en formato XML.

Una vez seleccionada una actualización para su descarga y a través de la información de la misma (*UpdateInfo*) deberemos proceder a descargar el fichero (en formato XML) que contiene la información necesaria (*DownloadInfo*) para comenzar a descargar las partes que formar nuestra actualización. Para gestionar los ficheros se ha creado una clase que representa el objeto en memoria:

- ***DownloadInfo*:** Representa la información de la descarga (representación en memoria del XML) una vez ya especificado que se desea descargar una actualización.

Otro de los componentes para la comunicación entre nuestra aplicación y ServiceWCF es la clase:

- ***WebServerRemote*:** Se comunica con *ServiceWCF* obteniendo la información de las actualizaciones/descargas: *UpdateInfo* y *DownloadInfo*.

Además se crearán una serie de objetos para gestionar todo lo referido a las descargas, objetos que gestionarán desde el comienzo de cada una (utilizando la información de *DownloadInfo*), hasta la gestión del sistema de reintentos, pasando por las opciones de pausa y reanudación. Para esta funcionalidad se han debido crear diversas clases:

- ***DownloadManager*:** Es la clase que contiene los métodos para la gestión de colas y eventos además de la funcionalidad básica de las descargas: “Añadir”, “Pausar” y “Resumir”.
- ***Downloader:*** Clase utilizada por *DownloadManager* para la gestión de las descargas parte a parte y en segundo plano.
- ***HttpRequest:*** Clase utilizada por *Downloader* para la descarga de cada parte vía http.

4.3.4 ServiceWCF

Componente necesario para la obtención de la información de las actualizaciones. Por ello, lo publicaremos en un servidor para poder atacarlo desde múltiples clientes. El servidor de ficheros será:

- ***ServiceWCF***, componente con los métodos necesarios para la obtención de los ficheros de actualización y descarga.

4.3.5 Controls

El siguiente componente estará formado por dos controles:

- ***Reloj del sistema:*** Control encargado de encapsular los procesos costosos del sistema mostrando un reloj en el momento de su ejecución y hasta su finalización. Estará formado por dos componentes, una clase estática con toda la lógica del control (*AppClock*) y una parte visual desarrollada como un "UserControl" (*EndlessClock*). El hecho de que la clase de control sea estática permitirá llamar al componente en cualquier momento, desde cualquier punto de nuestra aplicación y sin necesidad de instanciar ningún nuevo objeto.
- ***Visionado de "Release Notes":*** Componente que representará la información técnica de una aplicación. Para ello crearemos un diseño idéntico al del reloj teniendo dos componentes, una clase estática con la lógica (*UpdateRelease*) y otra con la interfaz (*ReleaseNotes*).

Otro de los objetos utilizado por estos dos controles; *UIElementAdorner*. *UIElementAdorner* estará basado en un concepto que Microsoft introdujo 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.

El control AdornerArea simula la funcionalidad del AdornerLayer. Se puede crear en cualquier ventana o control de usuario 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 4.10.

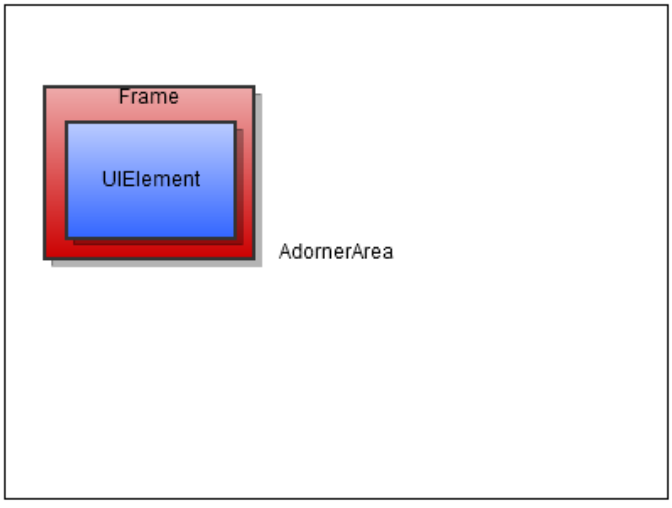

Figura 4.10: AdornerArea sobre una aplicación.

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 amplia el Adorner, éste le cambia el tamaño al UIElement dependiendo de la distancia que lo haya ampliado.

4.3.6 Sys

Para el arranque y parada de procesos y servicios del sistema necesitamos crear un componente encargado de gestionar esa funcionalidad que será necesaria para la instalación de una actualización que requiera de la parada o el arranque de un proceso o servicio. Para ello creamos una clase estática llamada *ProcessController,* con toda una serie de métodos de gestión basados en la utilización de dos clases más, *WinService* y *WinProcess*, que serán las que realmente contengan toda la lógica de arranque y parada.

4.3.7 SoftwareInstallation

Este componente ha sido diseñado para cubrir dos funcionalidades básicas en nuestro sistema. La primera es aportarnos una representación en forma de clase de los ficheros XML de actualización (*UpdateInfo*) que residen en el servidor, se utilizará por el componente *Network* para deserializar la información obtenida del *ServiceWCF.*

La segunda funcionalidad básica es la gestión integral de la instalación de la actualización una vez descargada. Como hemos comentado en puntos anteriores, una vez descargada, se procederá a su instalación y este proceso engloba una serie de clases. Las clases son:

- ***InstallUpdateInfo*:** Este objeto contiene la representación en forma de clase de la información necesaria para la instalación de una descarga, como que servicios y procesos a arrancar o detener antes y después de la instalación.
- ***TaskProcessor*:** Dado un objeto *InstallUpdateInfo* gestiona toda la instalación en el sistema, como es la lista de procesos que se deben detener o arrancar en cada momento. Para el arranque y parada de los procesos y servicios esta clase utiliza la clase *ProcessController* del componente *Sys*.
- ***UpdateInstaller*:** Las actualizaciones que descargaremos tienen extensión .zip (formato de compresión) ya que para realizar la instalación se necesita una información que muchas veces el instalador de por si no contiene. Por ello *UpdateInstaller* se encargará tanto de la descompresión de la descarga como del envío de las instrucciones de instalación (*InstallUpdateinfo)* a *TaskPoccessor* para su gestión.

4.3.8 Localization

Este componente implementará toda la función de localización permitiendo el cambio de lenguaje en tiempo de ejecución y soportando cualquier tipo de idioma (incluidos aquellos que requieren de fuentes UNICODE como el japonés o el ruso). También nos dará soporte para conocer

como representar variables de globalización, como son la monética, el formato de las fechas y el formateo de las cantidades. Las clases encargadas de estas funcionalidades son:

- ***GlobalizationData:*** Clase que implementará el formateo de valores numéricos, monético y de fecha en función de la cultura seleccionada. La cultura será un código es-ES donde las dos primeras letras indican el idioma "es" español y las dos siguientes "ES" indican el país, España.
- ***ApplicationGlobalizationData:*** Se encargará de establecer el idioma de la aplicación que estamos utilizando (*LiveUpdate*).
- ***SystemGlobalizationData:*** Se encargará de establecer el idioma del sistema (*Kiosk Gifts*).

Para aclarar por qué disponemos de dos lenguajes que pueden ser diferentes, para aplicación o sistema, podemos poner un ejemplo en el cual un cliente de Inglaterra dispone de un equipo con *Kiosk Gifts* con el idioma del sistema en inglés, pero tiene clientes de habla no inglesa que desean poder acceder en su propio idioma a las aplicaciones.

Otro de los puntos que implementará este componente es la traducción de las aplicaciones en pantalla en tiempo de ejecución. Para ello se implementarán los siguientes objetos:

- ***Translator:*** Clase encargada de obtener los mensajes traducidos para cada una de las claves o mensajes de la aplicación. Este objeto deberá ser inicializado con la ruta donde podemos encontrar los ficheros que contienen las claves y valores de traducción y con el idioma que deseamos para el aplicativo. Internamente se encargará de devolver a cada una de las claves su valor correspondiente. El sistema podrá funcionar de dos modos; el primero, usando *GlobalMsg* (mensajes globales del sistema) donde se pedirá al traductor que mensajes se desean traducir. El segundo será recorriendo los controles visuales y estableciendo el valor correspondiente de traducción a cada uno de ellos. En el caso de nuestra aplicación utilizaremos el primer método.
- ***GlobalMsg:*** Clase que contendrá la información de una clave y su correspondiente valor de traducción (ejemplo de *globalMsg* clave: btnGuardarDownload y valor: "Guardar descarga en disco").
- ***TranslatorStorage:*** Clase que se encargará de guardar a modo de *buffer* todos los mensajes de la aplicación para el idioma seleccionado. De ese modo, no se realizarán operaciones redundantes a disco en busca de los mensajes de traducción.

4.3.9 Skins

Proporcionará a la aplicación una definición de los diferentes estilos, donde cada estilo será la representación de un objeto en pantalla, definiendo cada una de sus propiedades y permitiendo modificar enteramente su presentación (color, fuente). Los estilos de WPF serían "equivalentes" a la relación entre un css y el html, siendo esta última relación más limitada.

Otro de los contenidos de este módulo será la definición de *templates* con los que podemos reemplazar la apariencia de cualquier control por algo completamente diferente, a través de un *Control Template*.

Cada control tiene un *template* por defecto, que le da la apariencia básica. El *template* está definido en cada control en la *Dependency Property Template*. Con un *Control Template* podemos reemplazar totalmente el árbol visual de un control.

Por ejemplo, un botón tiene el siguiente árbol visual (Fig. 4.11):

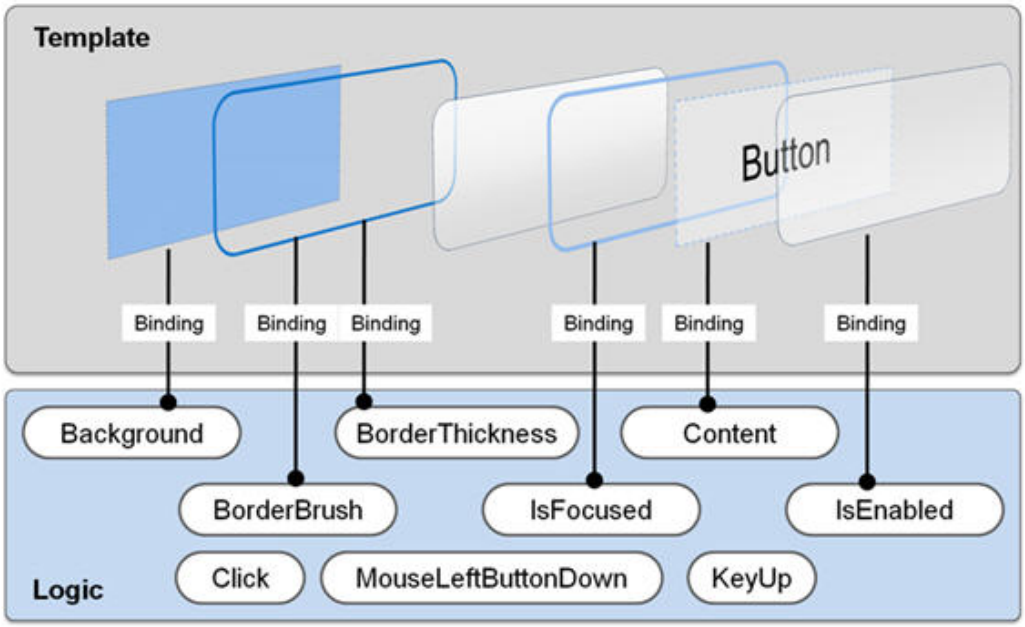

Figura 4.11: Diagrama donde se muestra la relación entre la parte visual y la lógica.

Este componente contendrá los siguientes ficheros de definiciones:

- ***WindowStyles:*** Contiene las definiciones del estilo básico de la pantalla principal que deben seguir todas nuestras aplicaciones.
- ***BasicResourcesDictionary*:** Contiene las definiciones de fuente utilizada, tamaño de la misma y colores utilizados por controles y textos.
- ***CollectionDictionary:*** Estilo para algunos controles: scroll y Listview.
- ***ControlsDictionary:*** Estilo y templates para controles básicos: buttons, images, borders y textblocks.

- ***SkinsDictionary:*** Se encarga de unificar los estilos antes especificados.

4.3.10 XmlHelpers

Componente para las tareas básicas de serialización/deserialización de clases en nuestro sistema. Su función será, a través de una clase, crear una representación física en forma de fichero en disco o la de crear a través del fichero físico una representación en forma de clase. Utilizaremos este componente en los objetos que contienen la información de la actualización:

- ***UpdateInfo*:** Objeto que contendrá la información básica de la descarga: nombre, tamaño y versión.
- ***DownloInfo*:** Visto en el módulo de Network.
- ***InstallUpdateInfo*:** Este objeto contendrá la representación en forma de clase de la información necesaria para la instalación de una descarga: servicios y procesos a arrancar o detener antes y después de la instalación.

4.3.11 InOut

Este componente contendrá la implementación de funciones de entrada/salida en nuestro sistema. Las principales funcionalidades para nuestro aplicativo serán las de partición y unión. La partición se realizará en el momento de la creación de la actualización. La unión se realizará una vez descargadas todas las partes. De esta operación se encargará:

- ***FileSplitter:*** Clase estática encargada de crear las partes de un fichero a partir de un tamaño definido. Además también se encargará de unir estas partes para volver a obtener el fichero inicial.

4.3.12 Config

El componente *Config* se creará para la gestión del fichero XML de configuración de la aplicación. Este fichero contendrá las rutas a las actualizaciones en disco, idioma de la aplicación y estilo que deseamos que se aplique.

4.3.13 Cryptography

Este componente contendrá la implementación de la función de encriptación del fichero de configuración para el módulo *Config*, además del cálculo y validación de CRC para la unión de las partes en la descarga dentro del módulo *InOut.* A continuación mostramos los objetos creados para este módulo:

- ***RijndaelCryptography:*** Clase encargada de la encriptación/desenciptación del fichero de configuración. Para ello utilizaremos el diseño creado por Microsoft en su framework de la plataforma .Net.
- ***GenericCRC:*** Clase que contiene la implementación de un CRC básico. La clase *GenericCRC* Es utilizada en la creación de las partes de la descarga, generando un CRC por parte y

guardándolo en el fichero XML con la información de la descarga. También se utiliza para una vez descargada una parte, volver a calcular su CRC y comprobar (utilizando la información de la descarga) si la parte ha sido correctamente descargada.

4.4 Planificación temporal

Una vez hecho el diseño de la aplicación se deben volver a lista las tareas que se deben llevar a cabo teniendo en cuenta las decisiones tomadas en el diseño. De esta manera se puede aproximar más esta planificación a lo que finalmente será el tiempo dedicado a desarrollar el proyecto. Al detallar las tareas también se explica más claramente en que punto se encuentra el proyecto durante su desarrollo.

4.4.1 Cambios en la planificación

Como consecuencia del diseño de la aplicación se ha detallado el conjunto de tareas que se deben llevar a cabo durante la implementación y se ha hecho una estimación más ajustada con la nueva distribución de tareas y utilizando como referencia para compararlas el estudio de viabilidad. Por ello, el tiempo de realización del proyecto se ha ampliado y el número de tareas y subtareas ha crecido.

A continuación enumeraremos las subtareas que hemos podido concretar para tareas ya existentes.

Nombre de tarea	Subtareas
Implementación interfaces de la aplicación	Pantalla principal
	Nueva barra de opciones
	Control para carga de Web
	Vista para Actualizaciones
	Vista para Descargas
	Vista para elementos instalados
	Vista para configuraciones
	Vista para Acerca de...
Implementación de Framework I	Módulo Network
	Módulo XMLHelper
	Módulo Skins
	Módulo Controles
	Módulo MVVM
Implementación de Framework II	Módulo Sys
	Módulo InOut
	Módulo Config
	Módulo Service WCF
	Módulo Cryptography

Al realizarse un estudio más profundo en el diseño de la aplicación hemos podido identificar todos los módulos necesarios para la misma, sobretodo en la implementación del Framework. También debemos destacar que el proceso de creación de interfaces y controles es la parte que más variación ha sufrido después de realizar el diseño de la aplicación, esto se debe sobre todo a la cantidad de controles que finalmente se crearán y la creación de estilos para cada una de las partes que forman la interfaz gráfica. Debido a que la implementación de interfaces se podrá realizar en paralelo con otras tareas planificadas la planificación no sufrirá grandes demoras (5 horas) y se podrán cumplir los plazos establecidos en el estudio de viabilidad del proyecto.

5 IMPLEMENTACIÓN

A continuación mostraremos el resultado final de la aplicación para cada uno de los módulos principales del sistema definidos en el análisis, además veremos partes internas (detalles) de nuestra implementación de la aplicación.

5.1 Interfaz del gestor de descargas

Nuestra nueva interfaz dispone de un diseño más moderno y funcional, además del soporte para una web corporativa de información al cliente.

Disponemos de una nueva barra de opciones (Fig. 5.1) que separa toda la funcionalidad en apartados de un modo lógico y que resulta más atractiva para el usuario.

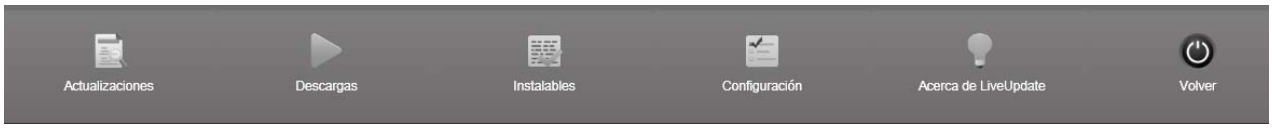

Figura 5.1: Nueva barra de opción.

El resultado final de la interfaz de la pantalla principal es (Fig. 5.2):

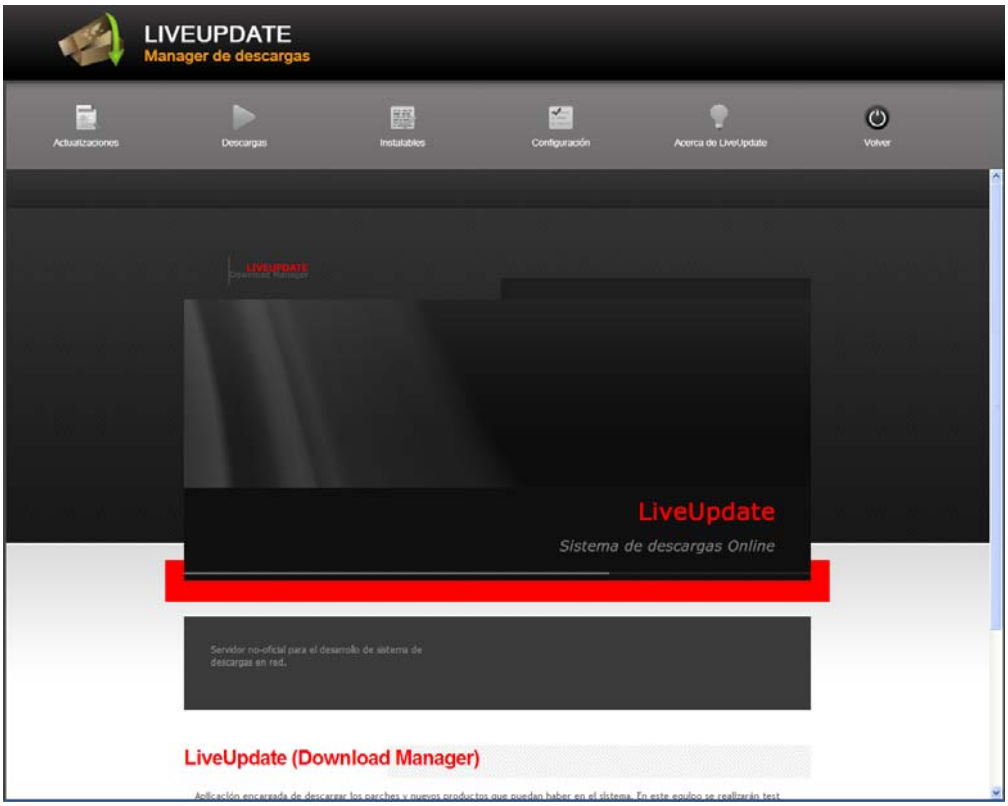

Figura 5.2: Nuevo sistema de descargas LiveUpdate.

5.2 Obtención de la información de las actualizaciones

Para la nueva pantalla de obtención de actualizaciones, hemos creado una serie de controles y opciones que permiten gestionar todos los aspectos de la actualización.

El primero de los controles será la lista con las actualizaciones, este control contiene toda la información necesaria para cada una de las actualizaciones.

Cada nueva actualización es mostrada con un control que aporta al usuario final información de la misma: nombre, descripción, tamaño, fecha de creación y notas de la versión (Fig. 5.3).

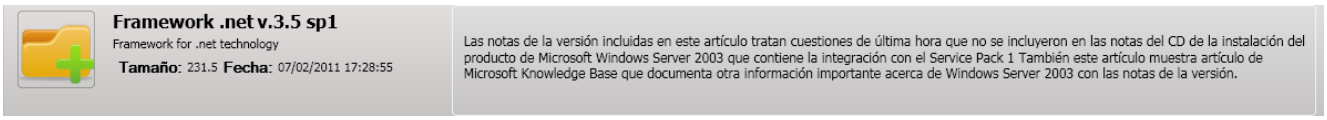

Figura 5.3: Control con la información de la descarga.

Debido a que el espacio del que disponemos es limitado hemos creado un control para mostrar el contenido completo de las notas de la versión. Este control (Fig. 5.4) dispone de un *scroll* (tipo cinético) para un visionado más cómodo desde una pantalla táctil.

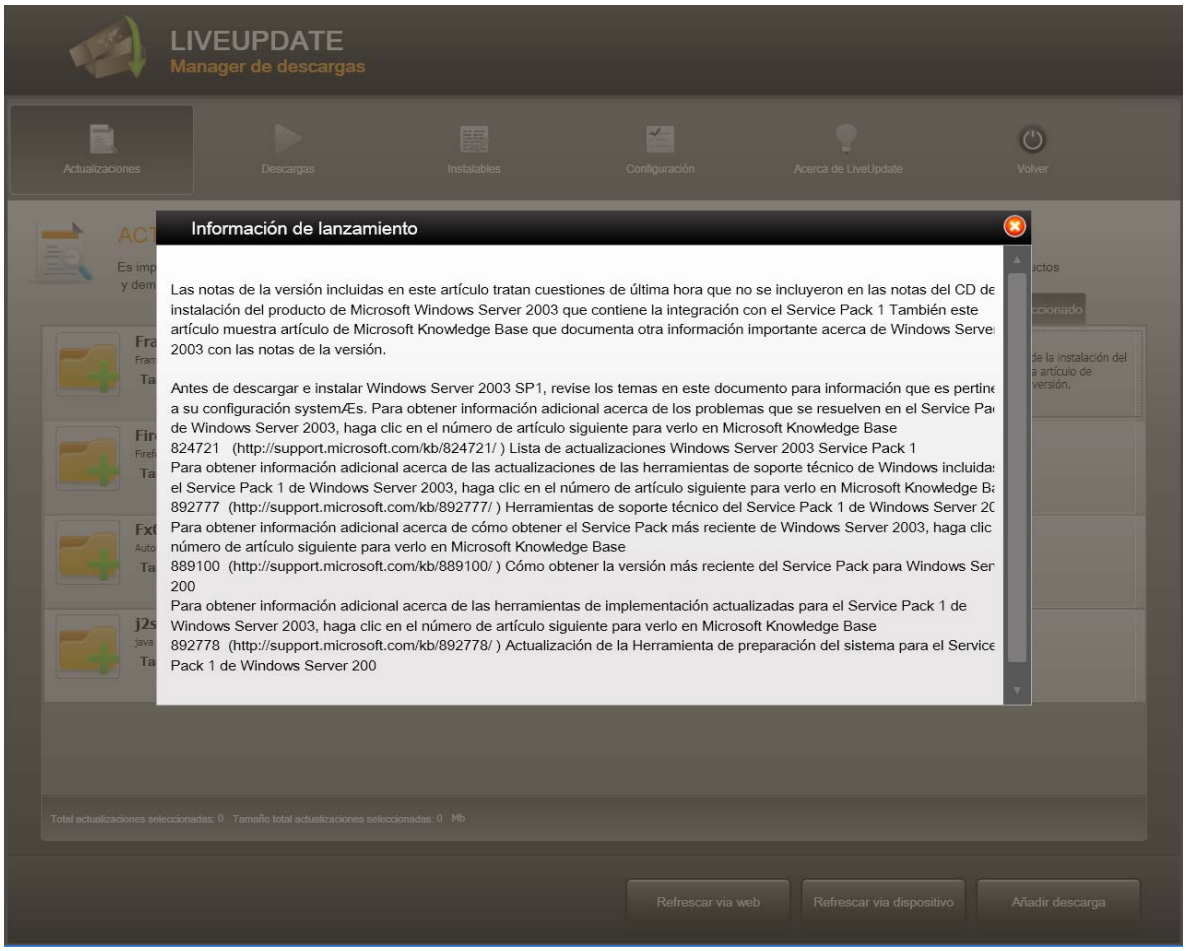

Figura 5.4: Control que muestra las notas de versión completas.

Para los procesos que pueden llevar tiempo al sistema, como obtener la información de las actualizaciones disponibles vía web o dispositivo, o el añadir actualizaciones a la lista de elementos a descargar se ha implementado un control en forma de reloj para aportar al usuario información de que el proceso se encuentra en ejecución.

A continuación mostraremos el aspecto final del control (Fig. 5.5):

Figura 5.5: Control mostrado en los procesos más costoso en tiempo.

Finalmente, el aspecto general para la pantalla de obtención de la información de las actualizaciones (Fig. 5.6) es:

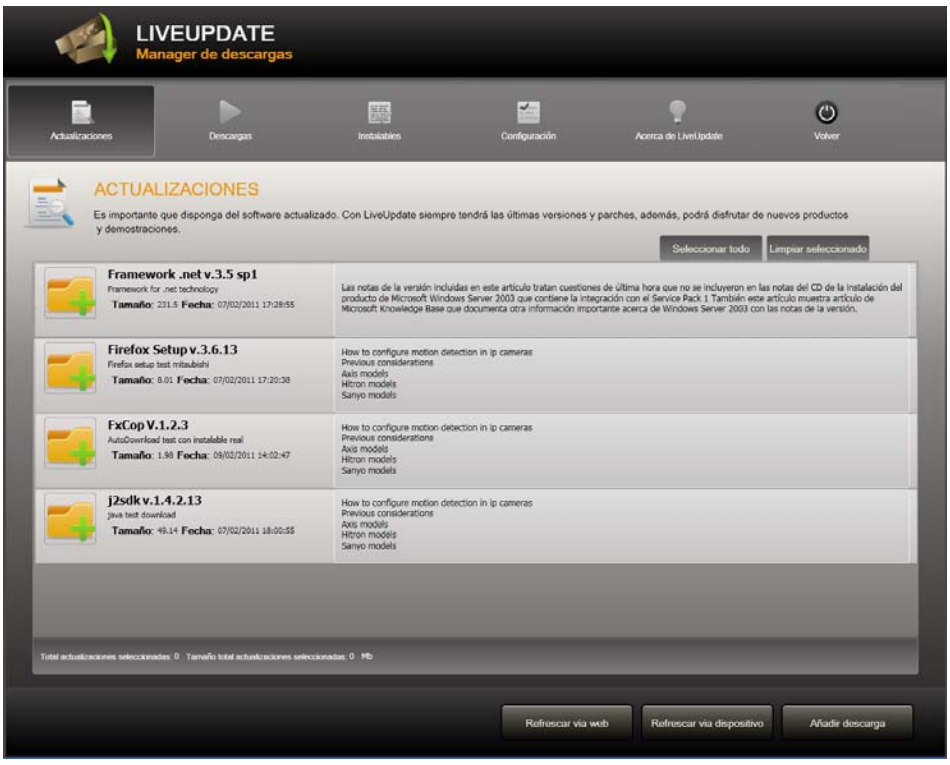

Figura 5.6: Pantalla de actualizaciones.

5.3 Gestión de las actualizaciones/parches en descarga

Para la gestión de las actualizaciones hemos creado una serie de controles y opciones que permiten gestionar todos los aspectos de la descarga.

El primero de los controles es la lista con las descargas. Este control contiene la información de las descargas en proceso. A continuación (Fig. 5.7) podemos ver el control:

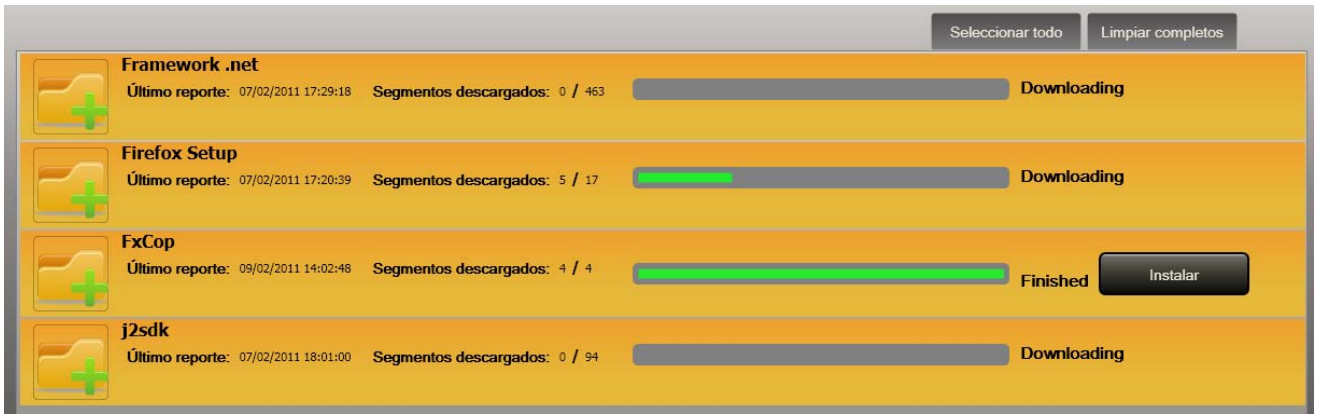

Figura 5.7: Lista con las actualizaciones en descarga.

Además de los nuevos controles (Fig. 5.8), también disponemos de nueva funcionalidad que nos permite pausar, reanudar o cancelar cualquier descarga en tiempo real.

Figura 5.8: Nuevas opciones para la descarga.

Finalmente, veremos una captura de pantalla (Fig. 5.9) donde se mostrará la estructura de la nueva pantalla gestión de descargas.

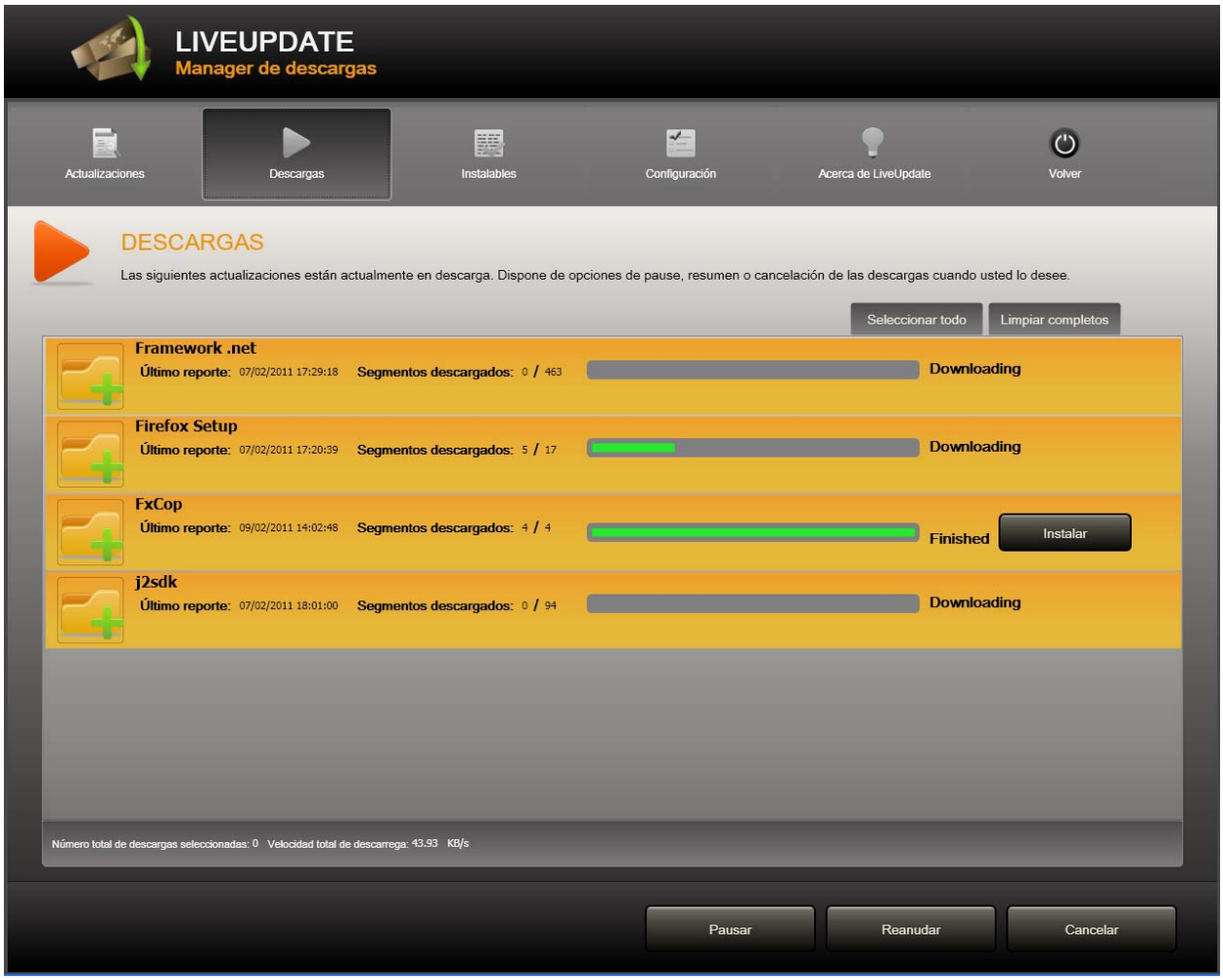

Figura 5.9: Pantalla de gestión de descargas.

Para ver más en profundidad como se ha implementado la funcionalidad de descarga utilizaremos los diagramas de clases que representan todas las partes que intervienen en la descarga: los objetos con la información de la descarga DownloadInfo (Fig. 5.10), WebServerRemote (Fig. 5.11) la clase encargada de obtener los ficheros con la información de las descargas y finalmente la estructura de clases encargadas de gestionar las descargas (Fig. 5.12).

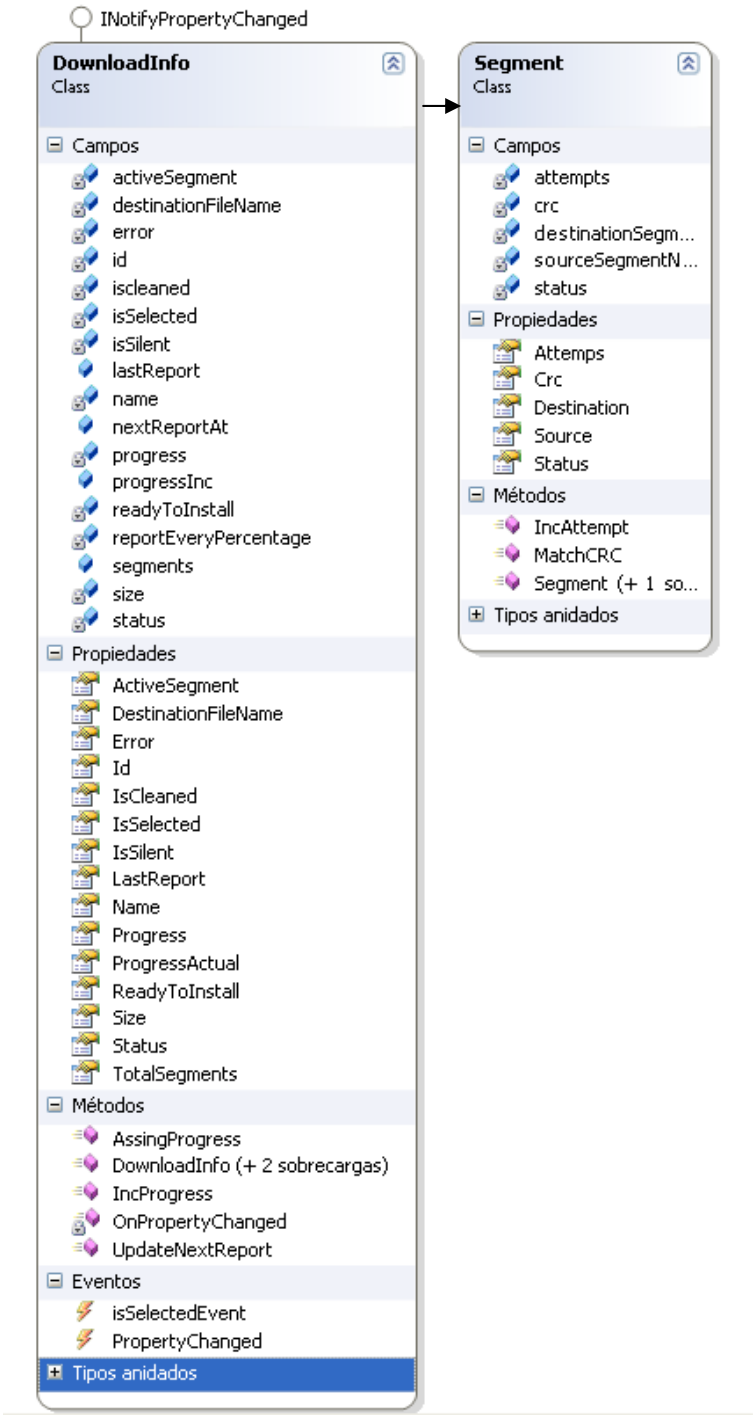

Figura 5.10: Diagrama de clases de la clase DownloadInfo.

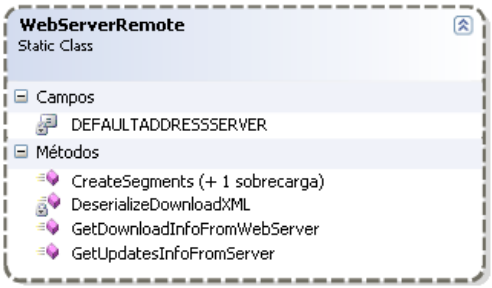

Figura 5.11: Diagrama de clases de la clase WebServerRemote.

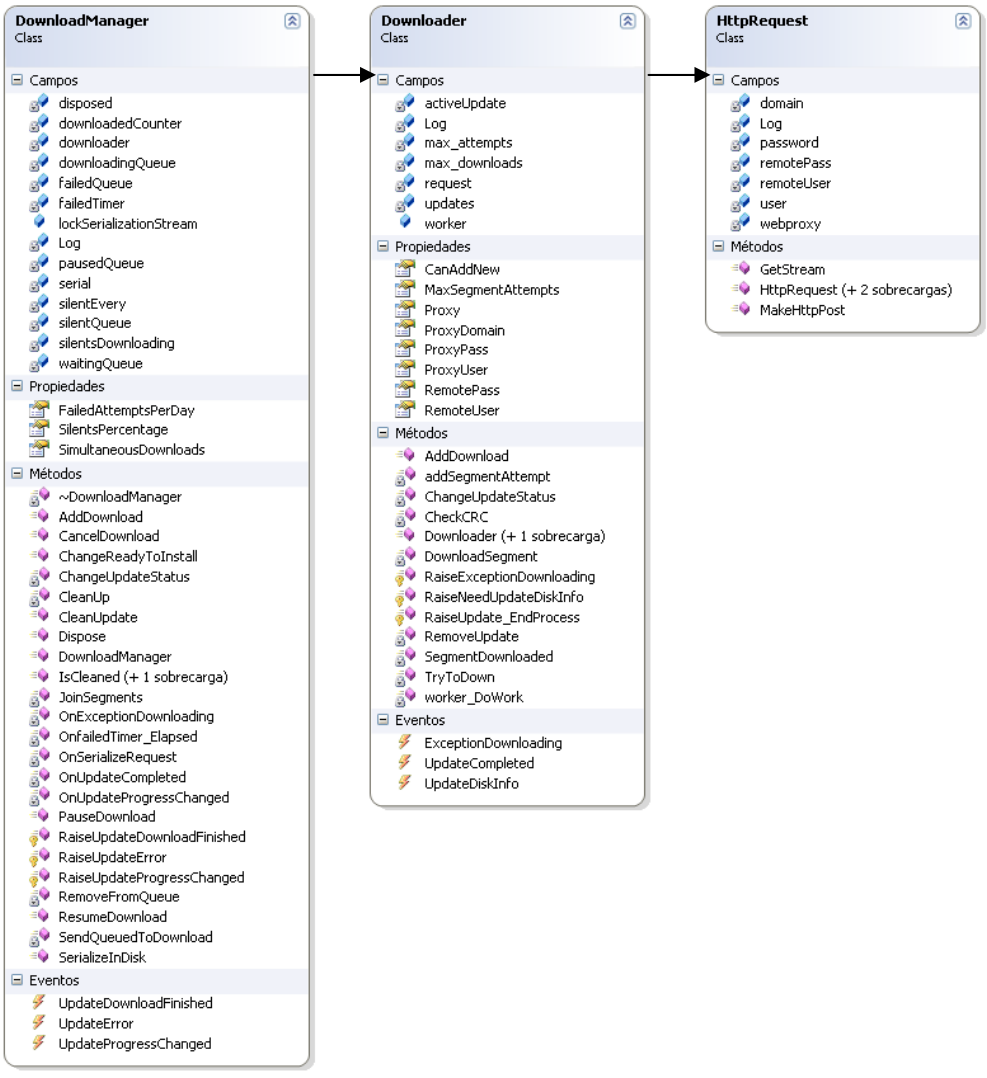

Figura 5.12: Diagrama de clases con las relaciones entre los objetos utilizados para la descarga.

A continuación veremos el flujo de instalación de nuestras actualizaciones una vez descargadas, mostraremos un diagrama de flujo de la lógica que sigue la aplicación (Fig. 5.13) y algunas de las clases que interviene en el proceso de instalación (Fig. 5.14).

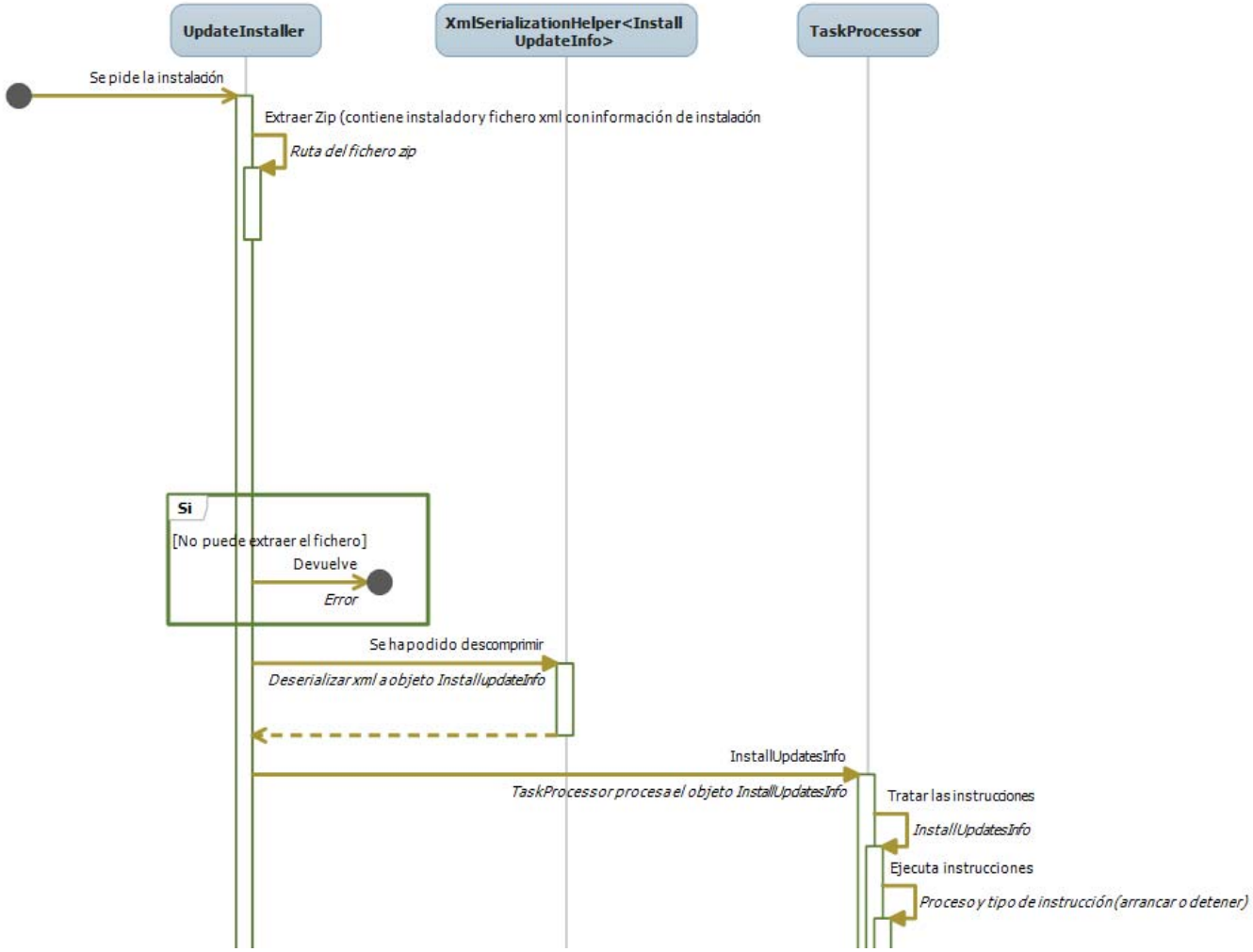

Figura 5.13: Diagrama donde se especifica el flujo de instalación de una descarga.

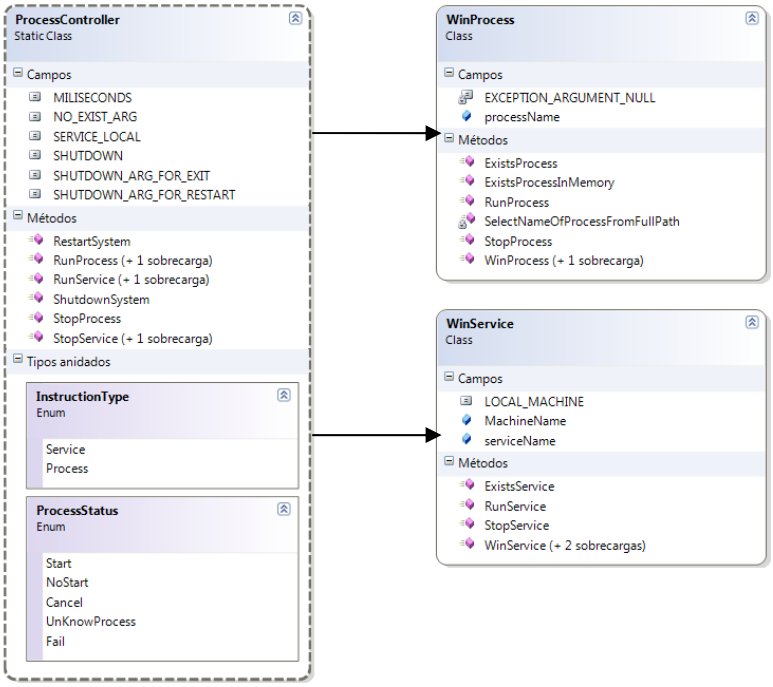

Figura 5.14: Diagrama de clases con la relación entre las clases del componente *sys*.

5.4 Visionado de las descargas finalizadas e instaladas

Para el módulo de las actualizaciones finalizadas hemos creado una serie de controles y opciones que nos permiten identificar la información de que descargas han sido correcta o incorrectamente instaladas.

Para la correcta presentación de los elementos descargados e instalados hemos creados dos listas: la de las descargas instaladas correctamente en el sistema y las que han registrado problemas en la instalación.

Para diferenciar que elementos, se ha definido un sistema de colores, teniendo la siguiente representación:

- Las descargas instaladas correctamente (en un color verde) (Fig. 5.15). También se permite acceder al archivo de la descarga en disco.

Figura 5.15: Control de descarga correcta.

- Las instaladas incorrectamente (en color rojo) (Fig. 5.16). También se permite acceder al archivo de la descarga o borrar la información.

Figura 5.16: Control de descarga incorrecta.

El resultado final de esta pantalla es el siguiente (Fig. 5.17):

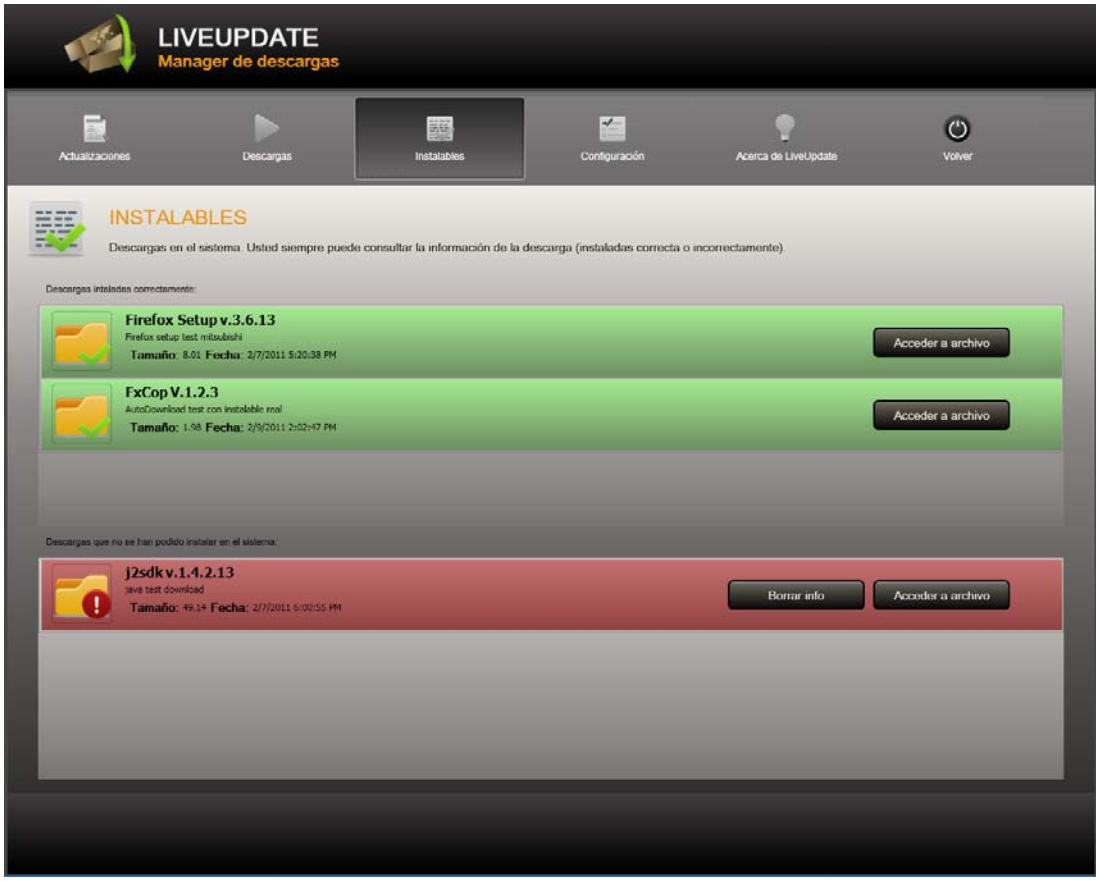

Figura 5.17: Pantalla de instalación final.

5.5 Configuración del sistema

Para el nuevo sistema se han creado tres nuevos parámetros de configuración. Estos nos permiten modificar el idioma y estilo de la aplicación, además de establecer si deseamos iniciar las descargas pausadas al iniciar el sistema.

El primero de los controles es un *combobox* que muestra los idiomas disponibles (Fig. 5.18). Cada uno de estos idiomas es mostrado en su idioma original (inglés como English, catalán como català).

Figura 5.18: Combo de configuración del idioma de la aplicación.

El segundo de los controles actúa como un *checkbox* pudiendo marcar o desmarcar la casilla en función de si estamos interesados en activar la posibilidad de iniciar las descargas pausadas después de reiniciar la aplicación (Fig. 5.19).

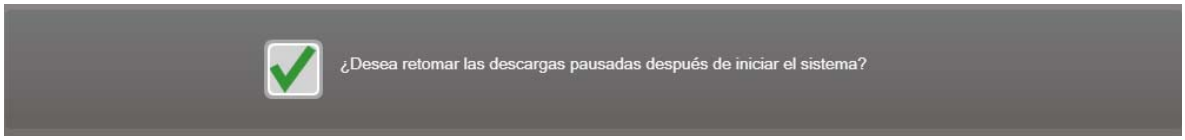

Figura 5.19: Checkbox de configuración.

Finalmente, el tercero de los controles permite modificar el estilo de la aplicación (Fig. 5.20). Para seleccionar que estilo deseamos dispondremos de un *Checkbox* con dos opciones: la primera (y por defecto) en “Estandard” y la segunda en “Alto contraste”. El control queda del siguiente modo:

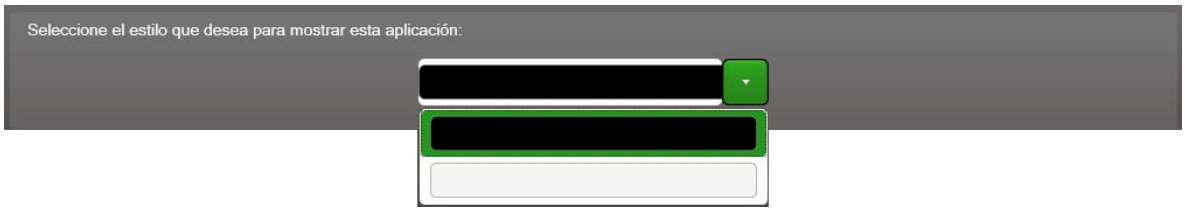

Figura 5.20: Checkbox de configuración.

Tenemos la siguiente captura para ver el resultado de cambiar de estilo a “Alto contraste” (Fig. 5.21):

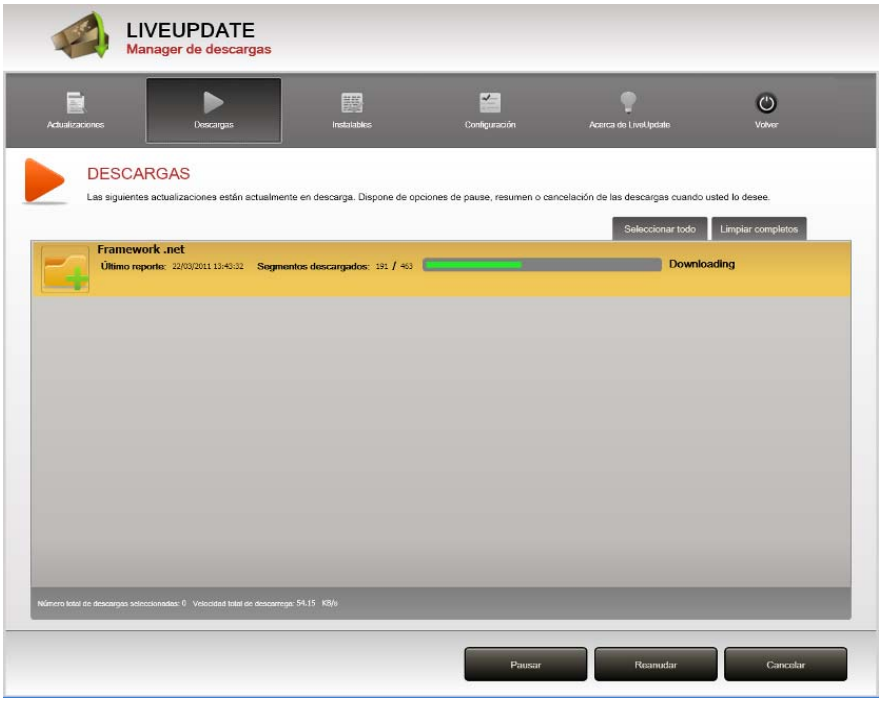

Figura 5.21: Pantalla de descargas en estilo “Alto Contraste”.

A continuación mostraremos el resultado final de la pantalla de configuración (Fig. 5.22).

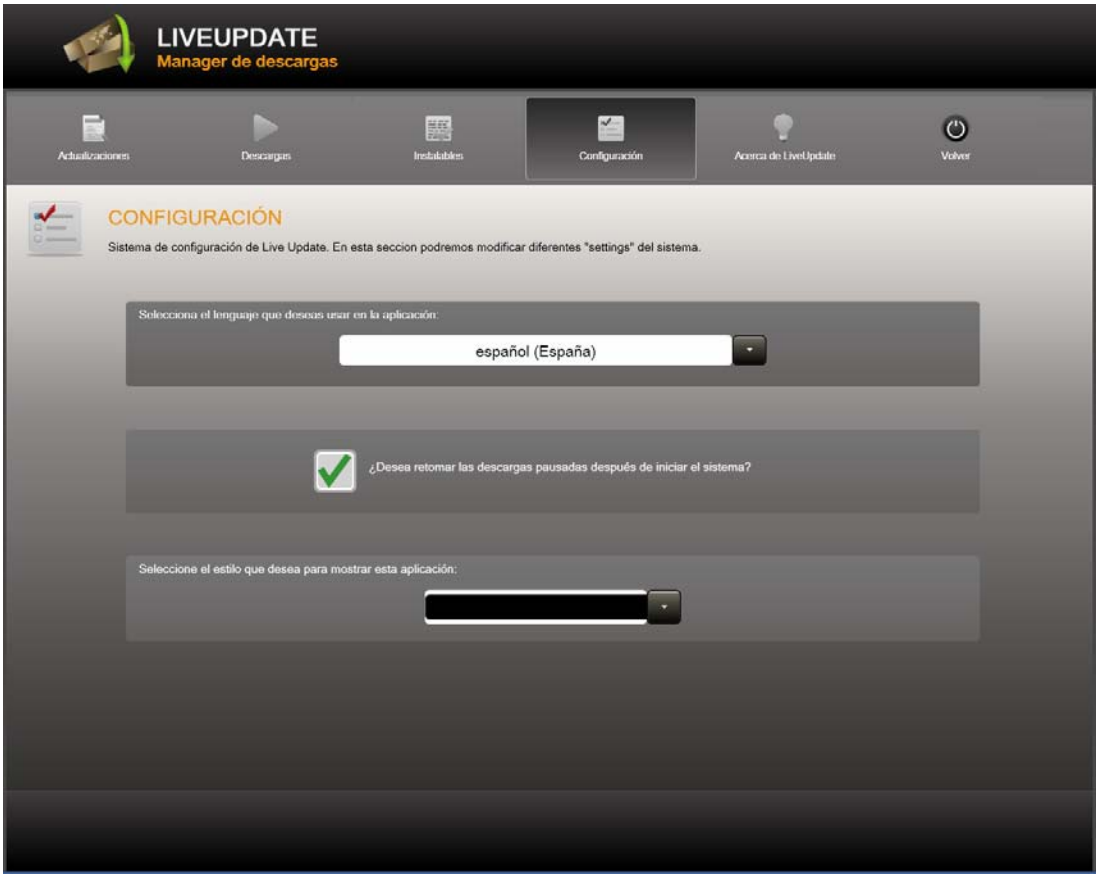

Figura 5.22: Pantalla de configuración.

Internamente, podemos destacar el flujo de traducción de la aplicación que está formado por tres implementaciones:

1. Obtención de la cultura a utilizar por la aplicación. A continuación mostraremos las clases que gestionan la cultura de nuestra aplicación (Fig. 5.23) y como se relacionan entre ellas.

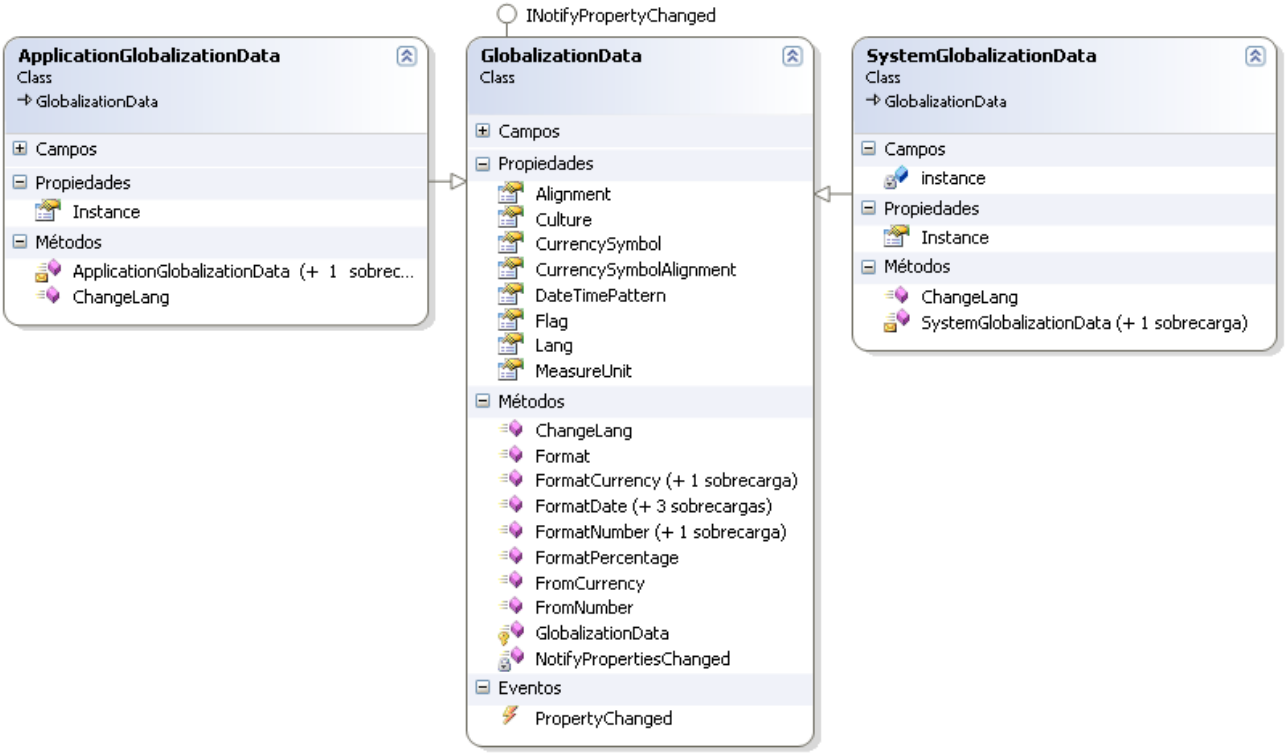

Figura 5.23: Diagrama donde se especifica la relación entre las clases con la información de la cultura del sistema o aplicación.

2. Validación de la existencia del fichero de traducciones y su posterior carga en *TranslatorStorage* (Fig. 5.24).

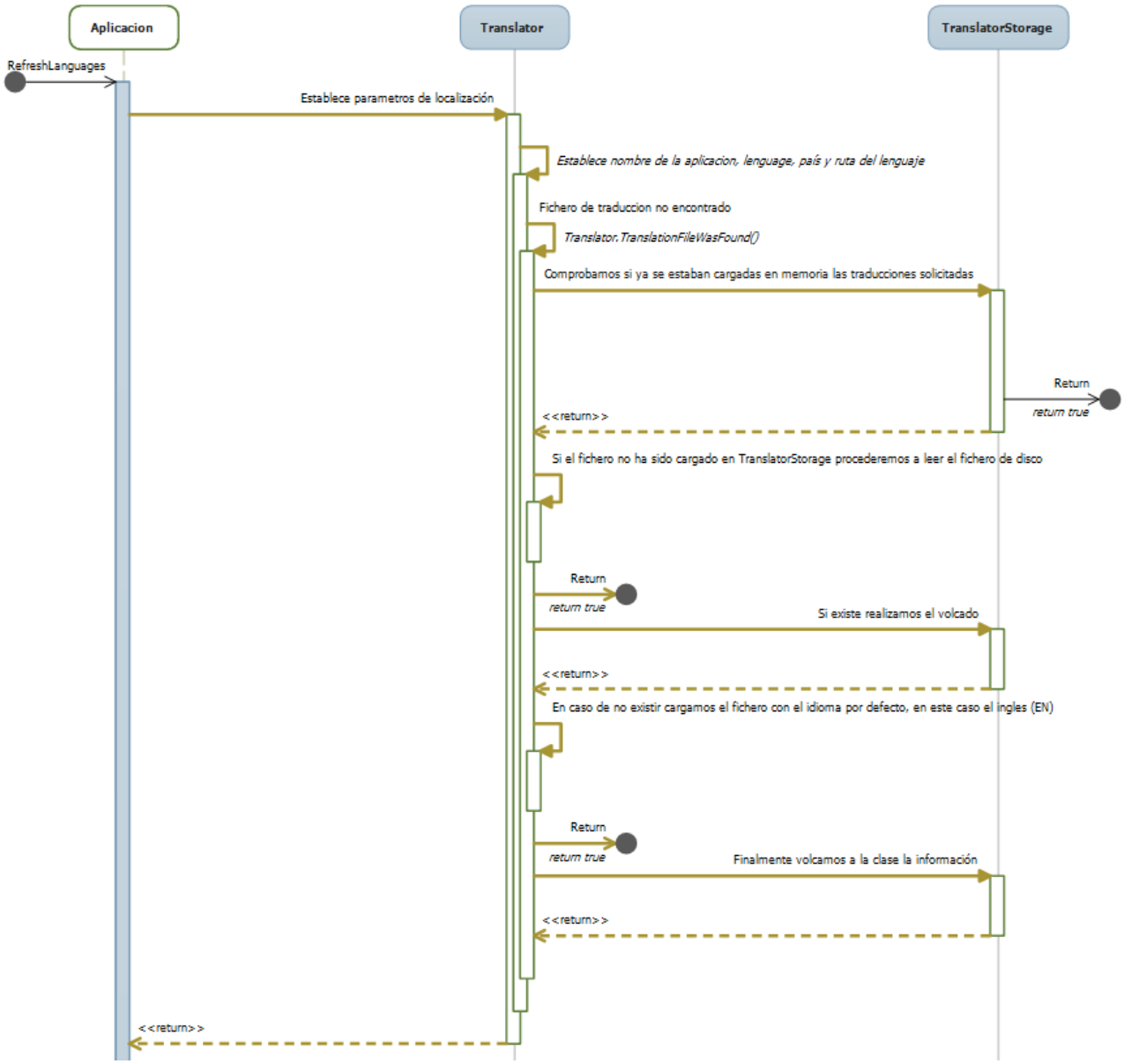

Figura 5.24: Diagrama con la secuencia para cargar un lenguaje en el sistema de localización.

3. Traducción de las claves. A continuación mostraremos el diagrama de secuencia (Fig. 5.25) con los pasos que se siguen en la traducción de las claves de una aplicación.

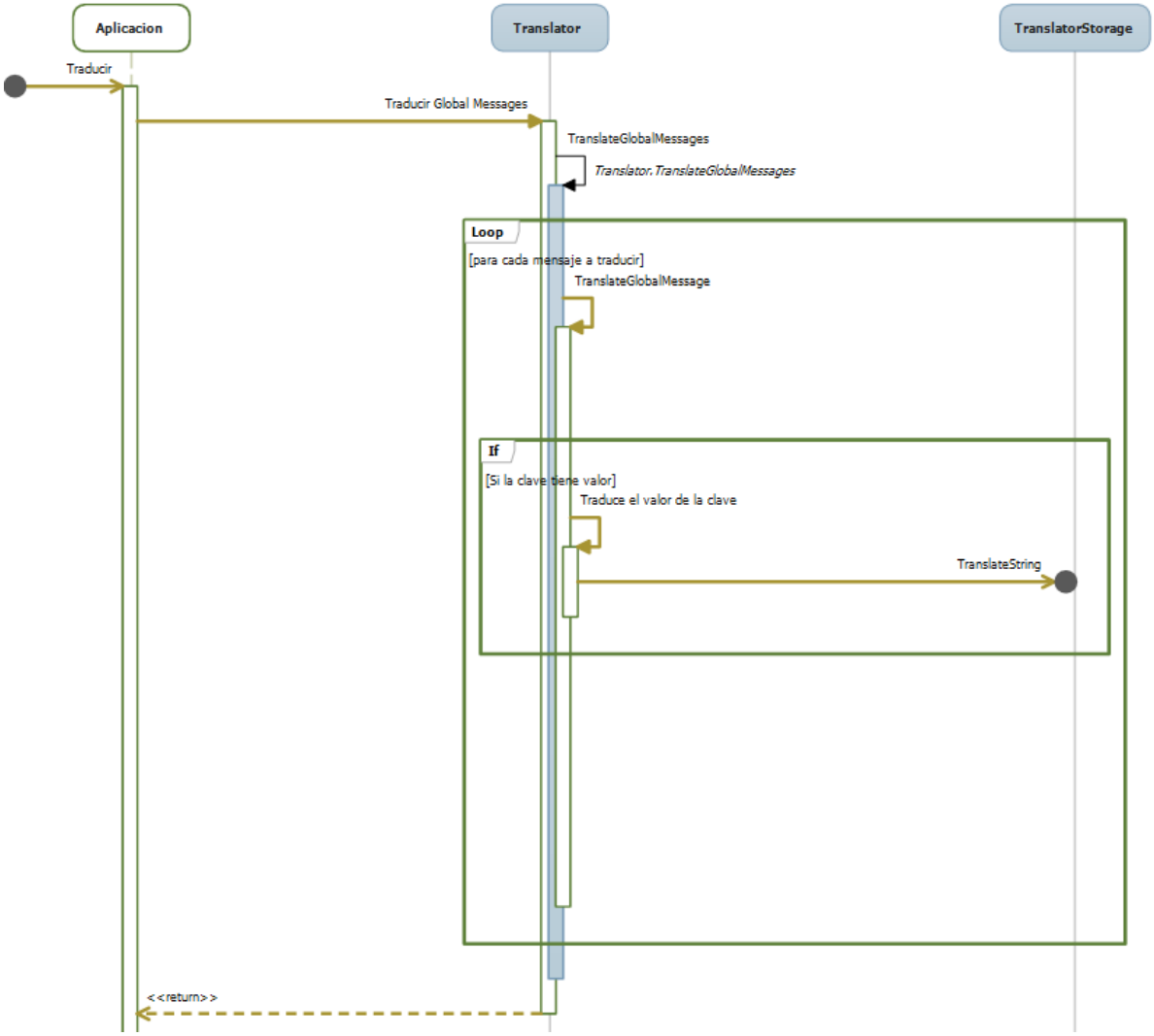

Figura 5.25: Diagrama de secuencia para la traducción de la claves.

5.6 Acerca de LiveUpdate

Esta nueva pantalla es uno de los elementos que se han añadido y que en el producto original no existía. La pantalla describe información básica de la aplicación: versión, autor, detalles del producto e información del copyright de la misma.

La nueva pantalla (Fig. 5.26) tiene el siguiente aspecto:

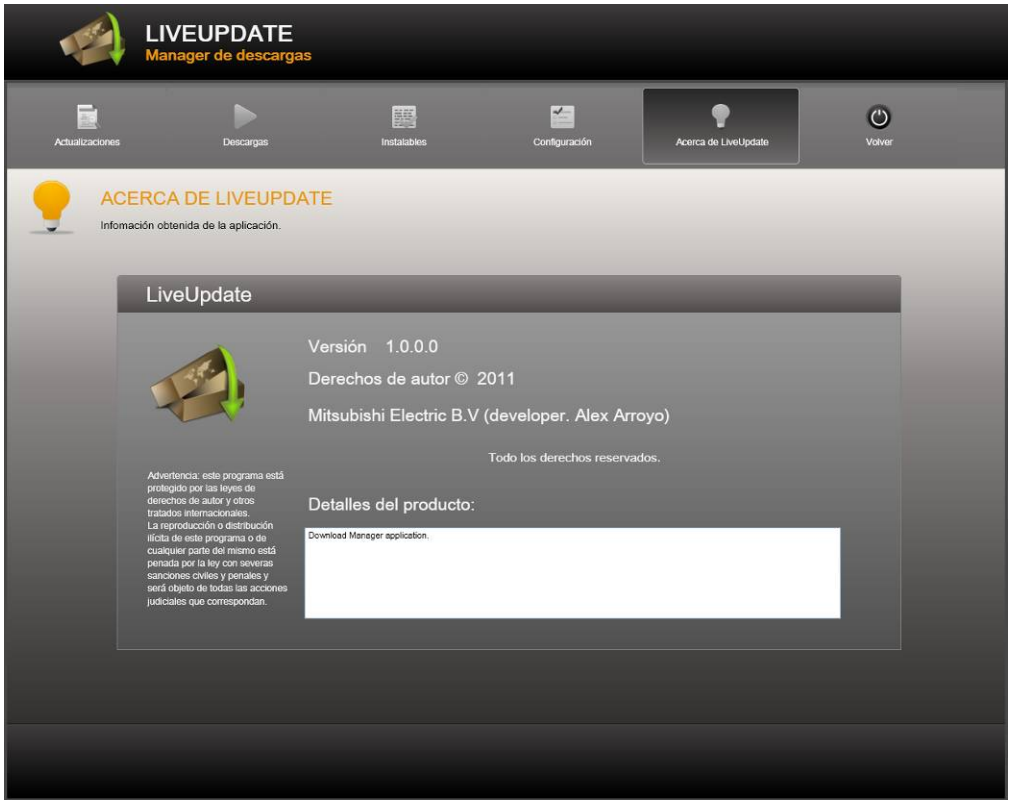

Figura 5.26: Pantalla de “acerca de”.

5.7 Nivel de consecución

En este apartado describiremos el nivel de consecución alcanzado con nuestra implementación. Estará estrechamente ligado con el cumplimiento de los requisitos y el seguimiento del diseño especificado en capítulos anteriores. Los puntos más importantes que se han alcanzado han sido:

- Conseguir una interfaz atractiva e intuitiva, donde se han podido separar todas las funcionalidades según su lógica (actualizaciones, descargas o instables).
- Un aplicativo mantenible y ampliable gracias a su diseño modular y a la implementación de patrones (MVVM) que mantienen separadas la lógica de las interfaces.

6 PRUEBAS

De la misma manera que se ha estructurado el proyecto en función de las capas de la aplicación, también se han hecho pruebas destinadas a comprobar cada nivel. Debido al concepto utilizado en nuestra aplicación era aconsejable realizar tanto pruebas independientes de las capas como globales para asegurar el correcto funcionamiento de nuestro aplicativo.

6.1 Pruebas de interfaz LiveUpate

Para realizar pruebas de toda la interfaz de *LiveUpdate* utilizaremos el aplicativo Visual Studio 2010. Visual Studio 2010 presenta un nuevo tipo de prueba: la prueba de interfaz de usuario codificada, que le permite crear pruebas de la interfaz de usuario automatizadas que luego se pueden agregar a un conjunto de aplicaciones de pruebas. Todo ello como parte de un conjunto de aplicaciones de prueba de comprobación de la compilación, lo que brinda al desarrollador comentarios inmediatos sobre cualquier error de la interfaz de usuario.

Para empezar creamos un proyecto de tipo “Test Project” y en el añadimos un componente “Coded UI Test” en el que establecemos el parámetro “Record Action”. Después se procede a iniciar *LiveUpdate* y ejecutar todas las acciones deseadas para el test, estas serán guardadas en el sistema. El paso siguiente será la generación del código fuente de test a través de las acciones capturadas.

A continuación establecimos el resultado que debía obtener nuestro test en su ejecución automática, para ello utilizamos el “localizador de control del UI”.

Repetiremos todo el proceso anterior tantas veces como test queremos definir en nuestro sistema.

Después de establecer test automáticos en todas las posibles acciones de la interfaz se consiguen identificar varios *bugs* en el sistema:

1. Se producen cortes en los mensajes de error en las descargas en algunos idiomas.
2. Problemas con el *scroll* de los elementos instalados, con cortes y desajustes.
3. *Crash* en el sistema cuando eliminamos elementos de la lista de actualizaciones desde un hilo secundario.

Soluciones a los problemas descritos:

1. Se crea un nuevo control (*ScalableFontSizeTextBlock*) el cual en función del tamaño que se dispone de texto modifica el tamaño de la fuente automáticamente.
2. Se revisan los componentes de *scroll* en *UserControls*.
3. Se modifica el código para eliminar elementos siempre desde el hilo principal.

6.2 Pruebas de Framework

Para las pruebas del *Framework* utilizaremos Visual Studio 2010. Desde visual Studio podemos creamos test unitarios para cada uno de los proyectos que forman el *Framework*.

Las pruebas unitarias se basan en la descomposición de un sistema complejo en otro más sencillo, es por esto que se separa en partes para poder comprender cada una individualmente.

Una prueba unitaria no es más que una clase con algún método que permite probar otro método, para evaluar el resultado se utilizan asserts (afirmaciones). Por tanto, nosotros especificamos el resultado que deseamos obtener para un método y las pruebas unitarias evalúan si el resultado de esa función es el esperado en un inicio.

A continuación la pantalla de creación de tests en Visual Studio 2010 (Fig. 6.1):

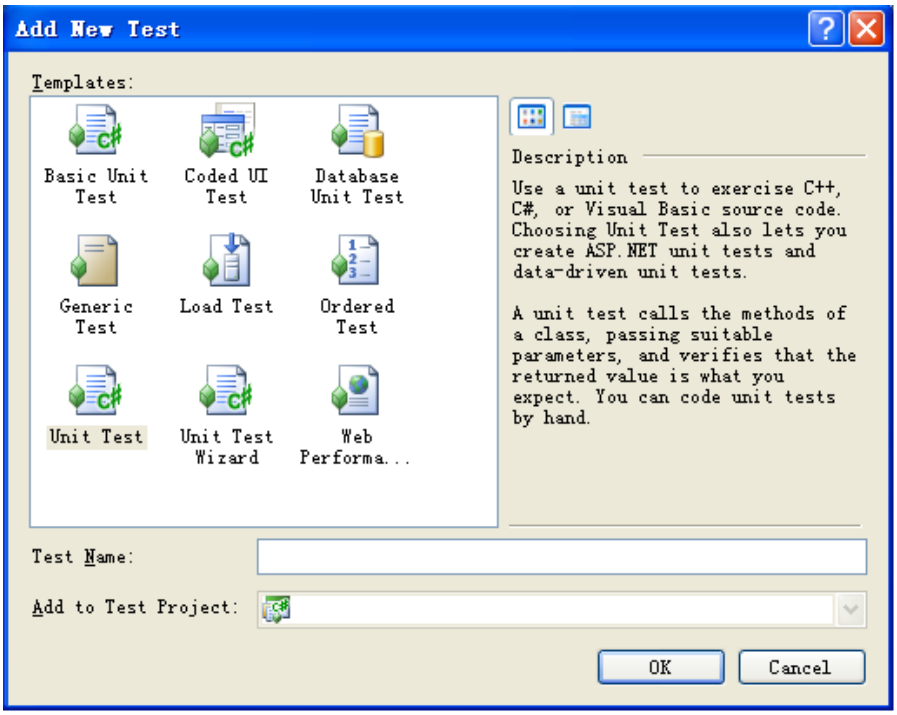

Figura 6.1: Pantalla para la creación de tests en Visual Studio 2010.

Después de generar test unitarios automatizados para cada una de los proyectos del *framework* pudimos identificar varias funcionalidades que no obtenían los resultados esperados:

1. El módulo de *Network* no identificaba correctamente el error producido en el sistema cuando se desconectaba el equipo del cliente de la red.
2. El sistema de reintentos en caso de fallo en la notificación del error en descarga se quedaba en un estado perpetuo de reintento.
3. En algunos casos se producían error en la unión de las partes para la generación del fichero original.
4. Problemas en el proceso de instalación de las actualizaciones cuando procesamos el xml con las instrucciones de instalación (InstallProcessInfo.xml).

Para solucionarlo se realizan las siguientes tareas:

1. Se controla la excepción producida al desconectar el equipo del cliente de la red.
2. Se soluciona errores en el sistema de reintentos en el sistema.
3. Se blinda el sistema de unión de partes.
4. Se solucionan el problema aparecido con el procesado de las instrucciones de pre-instalación).

Después de realizar los cambios en código podemos volver a pasar los test automáticos para verificar que el resultado de los test es el esperado.

6.3 Pruebas del Servicio Web (WCF en el Servidor)

Los servicios Web se pueden probar llamando a métodos Web desde pruebas unitarias. Probar servicios Web es muy similar a probar otros códigos mediante pruebas unitarias en las que se utilizan instrucciones Assert, y las pruebas generan la misma serie de resultados. Sin embargo, el espacio de nombres Microsoft.VisualStudio.TestTools.UnitTesting.Web de Visual Studio 2010 para Testers proporciona atributos y métodos específicamente para probar servicios Web.

La lista siguiente describe dos maneras de probar servicios Web con pruebas unitarias:

El servicio Web se ejecuta en un servidor Web activo: No hay ningún requisito especial para probar un servicio Web que se ejecuta en un servidor Web local o remoto, como IIS. Para hacerlo, se debe agregar una referencia Web y después, llamar a los métodos Web del servicio Web desde las pruebas unitarias, del mismo modo que éstas llamarían a los métodos de un programa que no fuese un servicio Web.

El servicio Web no está alojado en un servidor Web activo: Como se describe en “Probar un servicio Web localmente” (apartado 6.3.1), se puede probar un servicio Web que se ejecuta en un equipo local y no en un servidor Web, como IIS. Para ello, se utiliza un atributo que se proporciona en Herramientas para pruebas de Visual Studio 2010 para iniciar el servidor de desarrollo de ASP.NET. Esto crea un servidor temporal en el host local que aloja el servicio Web que se está probando.

6.3.1 Probar un servicio Web localmente

Este es el proceso para probar un servicio Web que se ejecuta en un equipo local pero no en IIS:

- Crear el servicio Web en el sistema de archivos local.
- Generar pruebas unitarias para el servicio Web de la manera estándar.
- Agregar el atributo AspNetDevelopmentServerAttribute a la prueba unitaria. Los argumentos para esta clase de atributo señalan al sitio del servicio Web y designan el servidor.
- Dentro de la prueba unitaria, agregar una llamada al método TryUrlRedirection para que el objeto de servicio Web señale al servidor correcto. Compruebe que devuelve el valor True y utilice una instrucción Assert para hacer que la prueba genere un error si falla la redirección.
- Llamar al servicio Web o utilizarlo de cualquier otra manera que se estime necesaria para probarlo completamente.

Gracias a estas pruebas pudimos identificar los siguientes *bugs*:

1. En algunos casos se producía un error en la obtención de los ficheros xml con la información de las actualizaciones debido a una mala deserialización de los ficheros.

Para solucionarlo se realizan las siguientes tareas:

2. Se modifica el método de deserialización para solucionar problemas aparecidos con ficheros XML de actualizaciones con codificaciones no estándar.

6.4 Pruebas funcionales

Una vez realizadas las pruebas de cada una de las partes y solucionados los problemas aparecidos se comienzan a realizar pruebas de sistema completo. Para ello, se ha probado la experiencia de simular 20 equipos con el sistema de actualizaciones *LiveUpdate* instalado durante 15 días. Para la simulación de los equipos se ha utilizado un sistema de creación de máquinas virtuales (que proporciona la empresa ATOS), este permite configurar todos los aspectos de una máquina. Iniciaremos el servidor con 5 descargas y aumentarán del orden de 25 descargas, con tamaños comprendidos entre 5 y 600 MB, en días alternos durante los 15 días de prueba.

Los resultados en simulación han resultado ser muy positivos, tanto por rendimiento como por estabilidad del sistema. El resumen del test sería:

- 16 equipos seguían activos y con todas las actualizaciones descargadas después de los 15 días del test.
- 4 equipos se habían reiniciado por motivos externos a *LiveUpdate* (fallos en otros aplicativos de *KioskGifts*) y proseguían descargando actualizaciones.

6.5 Pruebas reales

Se comienzan a realizar pruebas de sistema completo en entornos reales. Para ello, se ha probado la experiencia en 10 equipos con el sistema de actualizaciones *LiveUpdate* instalado durante 30 días. Estos equipos tendrán diferentes tipos de conexiones a internet:

- 2 Equipos con 1 MB de ADSL.
- 2 Equipos con 3 MB de ADSL.
- 2 equipos con 6 MB de ADSL.
- 2 Equipos con 10 MB de ADSL.
- 2 Equipos con 20 MB de ADSL.

El servidor de inicio contiene 15 descargas y se aumentarán del orden de 50 descargas, con tamaños comprendidos entre 5 y 1000 MB, en días alternos durante los 30 días de prueba.

Los resultados han resultado ser muy positivos, tanto por rendimiento como por estabilidad del sistema en diferentes escenarios. El resumen del test sería:

- 8 equipos seguían activos y con todas las actualizaciones descargadas después de los 30 días del test.
- 2 equipos, en este caso los equipos con conexión más lenta (1MB) todavía prosiguen con parte de las actualizaciones.

7 CONCLUSIONES

Una vez mostrado el resultado final del sistema y finalizadas las pruebas se realiza un balance con la planificación real del proyecto, observando las desviaciones con la planificación realizada en el apartado de "Diseño". Se realizará una valoración global de la experiencia y de posibles mejoras a realizar en un futuro sobre el sistema.

7.1 Seguimiento del proyecto

El seguimiento del proyecto permite controlar, a medida que se finalizan las tareas, en que momento se encuentra el desarrollo del proyecto. También es importante a la hora de valorar la planificación hecha al final de la fase de diseño mostrando todas las tareas con su duración real en relación con la planificación, de este modo se puede detectar en que fases el proyecto ha sufrido retrasos.

No ha habido desviaciones significativas en las primeras fases del proyecto (análisis, diseño o implementación), ya que en la planificación se preveía un coste importante de recursos debido a la complejidad del proyecto y al hecho de adquirir la formación necesaria para llevarlo a cabo durante su realización. Aun con eso, módulos muy extensos como Network, MVVM, el servicio WCF o los controles han requerido de más tiempo del planificado. Al realizarse las tareas en paralelo no han afectado al tiempo de finalización de la realización del proyecto, no superando el número de horas empleado el total estipulado por la asignatura del PFC.

Las desviaciones más significativas son las relacionadas con las fases de test que han obligado a rehacer partes de la implementación para resolver el problema de funcionamiento como se describe en el capítulo de pruebas. También ha resultado más costosa de lo planificado la redacción de la

documentación del proyecto debido, sobre todo, a la cantidad de partes que requerían de explicación y de esfuerzo de síntesis.

7.2 Experiencia y posibles mejoras

LiveUpdate que permite obtener actualizaciones de un sistema remoto o un dispositivo local, utiliza toda la funcionalidad del *Framework* que aporta modularidad y escalabilidad, además ha sido diseñado con algunos de los lenguajes más importantes actualmente (C# y WPF). Después de haber realizado el proyecto, el conjunto de componentes y haberlos probado, podemos extraer algunas conclusiones respecto a los resultados obtenidos y los objetivos iniciales. Por tanto, podemos concluir:

- WPF es una buena opción de conjunto de desarrollo de aplicaciones de escritorio gracias al marco de .NET, que aporta un conjunto de librerías profesionales y una comunidad muy activa. 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.
- 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.
- En una primera aproximación a WPF para desarrollar controles, para mostrar información tanto de las actualizaciones como de las descargas así como para realizar una interfaz atractiva en todos los sentidos, vemos que no existe ningún tipo de limitación, pero sería interesante investigar librerías de externas, como pueden ser las desarrolladas por empresas como Infragistics o VantagePoint.
- El *Framework* se puede aplicar a cualquier tipo de aplicación WPF o C#. Todo esto gracias a la versatilidad, escalabilidad y modulación que aportan los módulos de *MVVM, Mediator, Network o Controls*.

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.
- Investigación del campo de carga dinámica de servicios WCF para el módulo *Network*.
- Investigar si es posible mejorar la experiencia del programador con el diseñador utilizando 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.
- Investigar la posibilidad de disponer del servicio web en múltiples servidores.

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. Los resultados creemos que son buenos y con

tiempo y más gente involucrada, se puede conseguir un *Framework* más potente que de aplicaciones profesionales.

De manera autocrítica, el desarrollo de este proyecto ha requerido un esfuerzo muy grande pero los resultados obtenidos son muy positivos. A nivel personal me ha permitido aprender temas muy interesantes y actuales de la programación orientada a objetos, además he podido ver todas las fases que componen el desarrollo de un proyecto, el estudio de viabilidad, el análisis o la creación de juegos de pruebas. Finalmente, puedo asegurar que mi bagaje profesional ha crecido mientras desarrollaba el PFC y que ahora dispongo de más herramientas para afrontar cualquier nuevo proyecto que pueda surgir.

8 BIBLIOGRAFÍA Y REFERENCIAS

8.1 Libros

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

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

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

8.2 Referencias en internet

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/.

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.

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

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

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

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

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

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

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

ANEXO I: DIAGRAMAS DE CLASE (FRAMEWORK)

En este anexo mostraremos algunos de los más interesantes diagramas de clase de algunos módulos de *Framework*:

Diagramas del módulo Controls

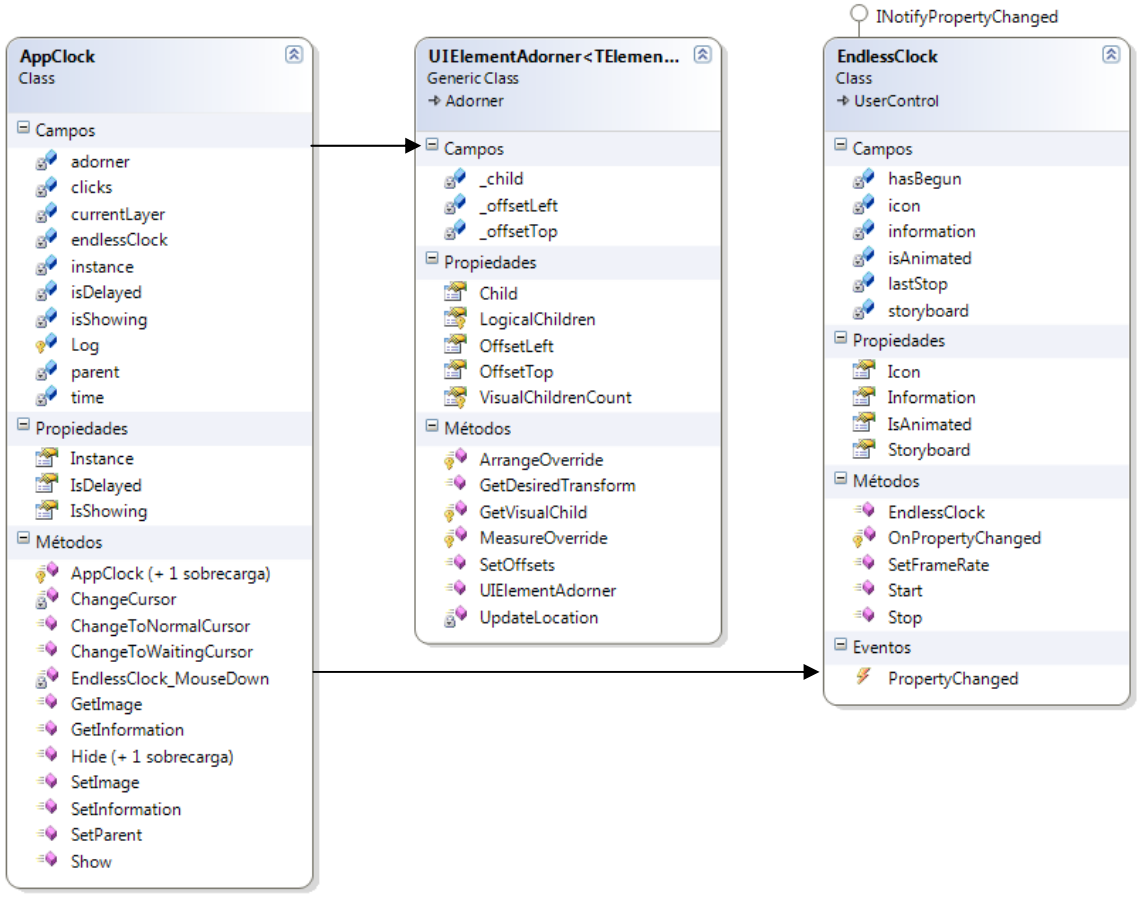

Figura I.1: Diagrama de clases con las relaciones entre los objetos utilizados en el control.

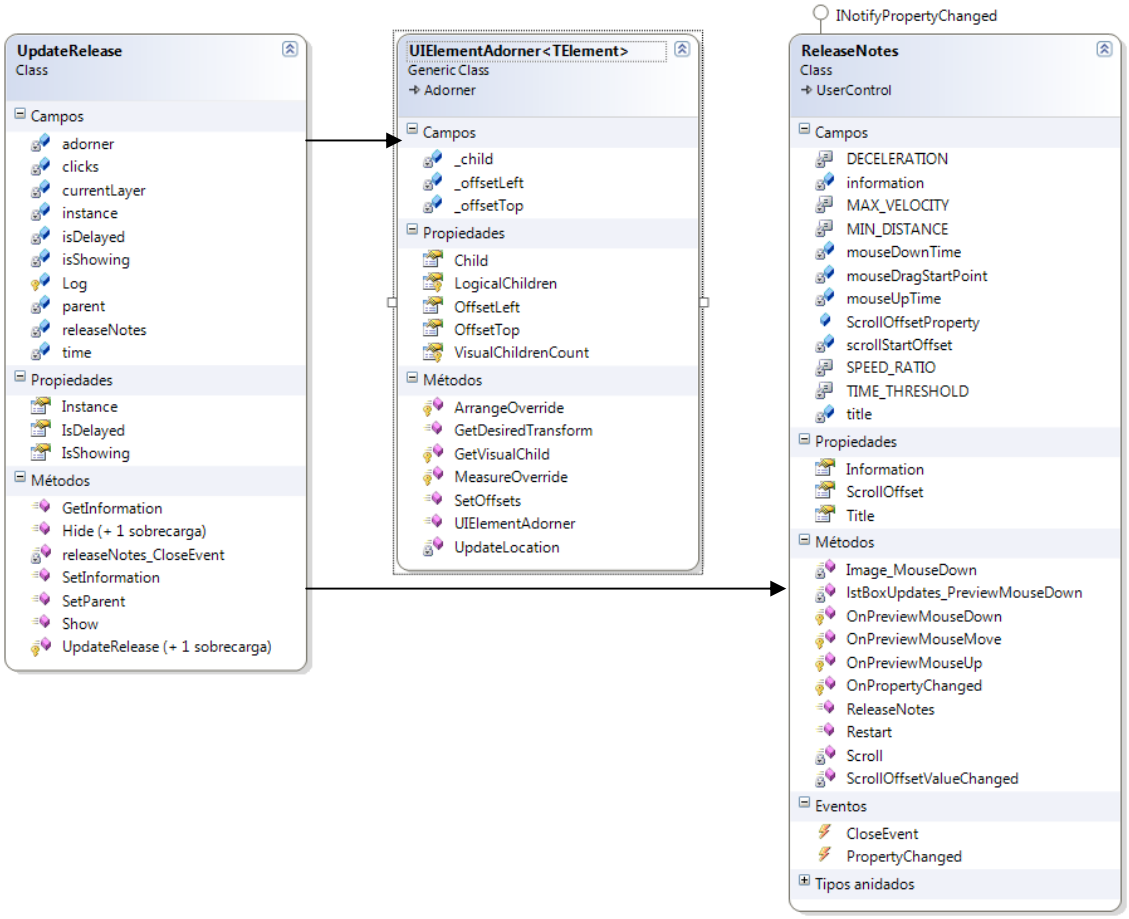

Figura I.II: Diagrama de clases con las relaciones entre los objetos utilizados en el control.

Diagramas del módulo Sys

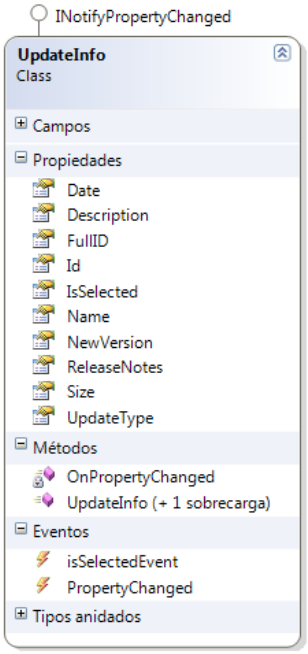

Figura I.III: Diagrama de clases de la clase UpdateInfo.

Diagramas del módulo XmlHelpers

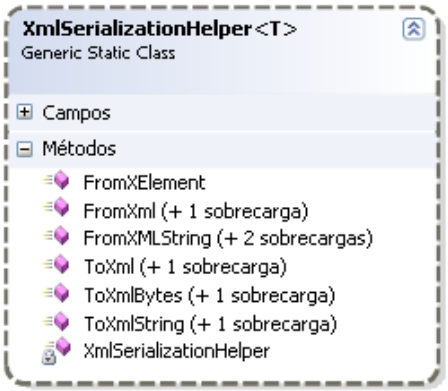

Figura I.IV: Diagrama donde se especifica los métodos del módulo.

Diagramas del módulo InOut

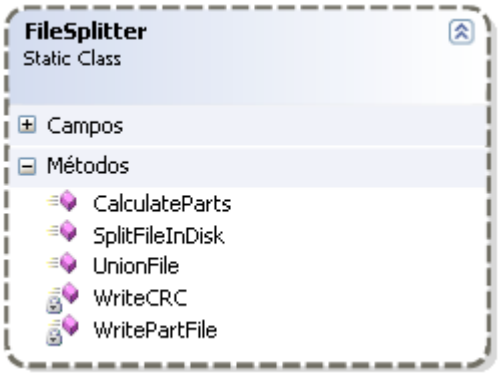

Figura I.V: Diagrama donde se especifican los métodos del módulo.

Diagramas del módulo Cryptography

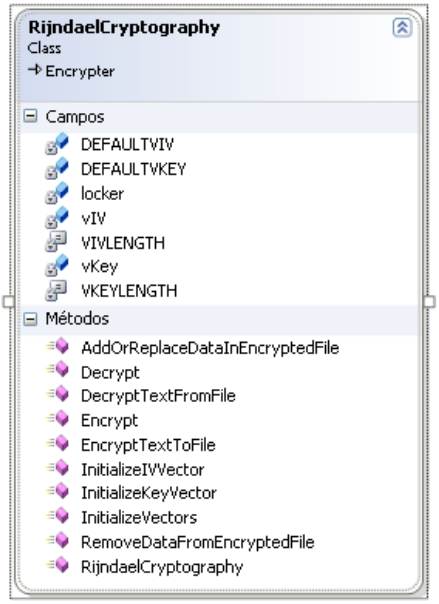

Figura I.VI: Diagrama donde se especifica los métodos de la clase.

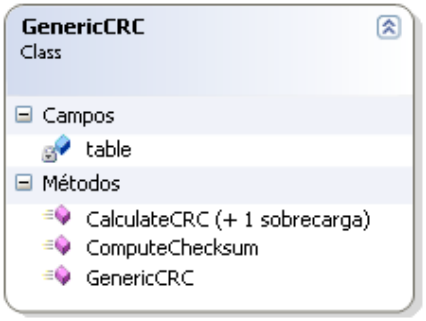

Figura I.VII: Diagrama donde se especifican los métodos de la clase.