Universidad de las Ciencias Informáticas
Facultad 7

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Título: Actualizador Automático

Autores: Anavel Delgado Murciano
Yoelkis Jarrosay Charadan

Tutor: Lic. Yasel Couce Sardiñas

Ciudad de La Habana, junio del 2010

"Año 52 de la Revolución"

RESUMEN

El presente trabajo se centra en el desarrollo de un sistema que facilite la gestión de actualizaciones automáticas para las soluciones informáticas desarrolladas en el Centro de Informática Médica.Su desarrollo está basado en tecnologías como: Framework.net 2.0 para facilitar la migración a plataformas libres haciendo uso de Mono 2.2, Nant que es una herramienta libre de código abierto para la automatización de procesos y sobre una arquitectura en capas, empleándose el estilo arquitectónico en tres capas. Se utilizó C# como lenguaje de programación que proporciona un acceso fácil a secuencias XML validadas por un esquema, en este caso el de Nant.

La aplicación desarrollada cuenta con una alta tolerancia a fallos que garantiza la estabilidad de los productos desplegados por el Centro de informática Médica. Mediante el engranaje de acciones sencillas se logran abarcar todo tipo de actualización que pueda necesitar una solución informática. La posibilidad de restablecer y deshacer cambios realizados por actualizaciones brinda seguridad a las aplicaciones y a la información manejada por estas, de forma que se garantice la extensión de la vida útil de las mismas. El sistema obtenido en esta investigación constituye un impacto positivo sobre la estrategia de comercialización y despliegue de productos del Centro de Informática Médica.

Palabras Claves

Actualizaciones automáticas, Informática para la Salud.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la actualidad, uno de los principales factores que influyen sobre la calidad de los servicios que brinda cualquier empresa u organización es, sin lugar a dudas, el control de la información. Esta surge producto de su interacción con la sociedad. El vertiginoso crecimiento de los volúmenes de datos que se originan dificulta su manipulación en gran medida y por ello en algunos casos, es necesario buscar nuevos métodos para administrarlos. Por lo mismo, a lo largo de la historia el hombre ha creado estrategias y medios de almacenamiento para lograr que su información permanezca organizada, segura y disponible. Sin embargo, no todas las prácticas puestas en marcha fueron suficientemente robustas. Con el advenimiento de las computadoras en el mundo, en la segunda mitad del siglo XX, fue que realmente se descubrió una forma más confiable de manipular la información: los programas computarizados.

Estos programas han ido evolucionando paulatinamente, hasta convertirse en sistemas informáticos, los cuales gozan de gran popularidad y aceptación en todos los sectores sociales. Han propiciado que se agilice el flujo de las actividades de las instituciones donde se aplican, mejorando su rendimiento y sirviendo de apoyo para la toma de decisiones, explotando al máximo las innovaciones tecnológicas con el objetivo de informatizar la sociedad. Con el devenir de estos sistemas se fueron disipando gradualmente los errores humanos que se cometen con los métodos tradicionales de gestión de la información.

La Informática es la ciencia que estudia los métodos, procesos y técnicas de manejo de la información. Esta ha tenido un desarrollo extraordinario expandiéndose a las disímiles ramas de la sociedad por lo que puede afirmarse que el futuro está estrechamente vinculado con las Tecnologías de la Informática y las Comunicaciones (TICs). La salud no es una esfera de la sociedad ajena a este adelanto tecnológico, que se inserta con el objetivo de mejorar el proceso de atención al paciente y disminuir la carga de trabajo para los trabajadores del sector. Estos trabajadores dedican mucho tiempo y esfuerzo en tramitar la información que se genera a partir de los procesos llevados a cabo en los distintos niveles de salud. De la confiabilidad y autenticidad de los datos depende la toma de decisiones de los médicos y a su vez la calidad de la atención propiciada al paciente.

Cuba, actualmente, cuenta con la Universidad de las Ciencias Informáticas (UCI); un centro estudiantil que además de la amplia gama de estudio que maneja, logra insertar a sus estudiantes a proyectos productivos en los que se desarrollan software para las distintas esferas de la sociedad. En la Facultad 7 específicamente, se encuentra ubicado el Centro de Informática Médica (CESIM) donde se crean aplicaciones informáticas destinadas a este sector. Las mismas son liberadas bajo la marca comercial alas. Los sistemas alas RIS, alas HIS y alas PACS son algunas de estas soluciones que aumentan y perfeccionan sus funcionalidades paulatinamente debido a la gran demanda de estos productos.

Con la inserción de la Informática en los distintos sectores de la sociedad, muchos han sido los problemas resueltos, pero así mismo otros han tenido su origen en ello. Ante la aparición de los sistemas informáticos surgen nuevas interrogantes y problemáticas a resolver que hoy preocupan a la comunidad desarrolladora. El mantenimiento es una parte fundamental del ciclo de vida de cualquier aplicación, entonces, ¿cómo brindar mecanismos y herramientas dirigidas a mejorar el soporte técnico y mantenimiento para todos los usuarios de las aplicaciones?; es hoy tema de interés y debate para ingenieros, investigadores y especialistas. Surge entonces la necesidad de definir un proceso para la actualización de los sistemas informáticos porque, simplemente, hay cosas que no se pueden predecir cuando se implementa un software. Para ello se han definido cinco elementos básicos que describen la esencia del problema.

Ocurrencia de incidencias: No existe un equipo experto que se encargue de solucionar cualquier problema o incidencia, lo cual hace el funcionamiento de las aplicaciones crítico para el negocio.

Demoras en las respuestas: No se cuenta con un nivel de respuesta inmediato, si surge cualquier problema, no hay nadie al lado del cliente para solucionarlo.

Estacionamiento de funcionalidades: No existe un mecanismo que permita ampliar las funcionalidades de las aplicaciones de manera progresiva y sin afectar a los clientes.

Malgasto de recursos: Los clientes y proveedores emplean gran parte de sus recursos en el mantenimiento de las aplicaciones.

Dependencia de servicios: No existe un servicio externo del lado del cliente que garantice un funcionamiento correcto de sus aplicaciones.

Los actualizadores automáticos son una respuesta a la interrogante antes planteada, pues ofrecen un servicio flexible de soporte y mantenimiento que garantiza el correcto funcionamiento de las aplicaciones, cubre las necesidades de evolución y asegura la calidad del servicio.

En esencia, un actualizador automático es una aplicación que permite al usuario comprobar si hay una nueva versión de su sistema. Si ésta existe, se brinda la posibilidad de descarga e instalación de la versión que le sigue a la que está en uso. De esta manera, se evita la creación de un módulo de actualización propio para cada aplicación o realizar el proceso manualmente [1]. Las tareas comunes descritas por un actualizador son las de descarga, copias de seguridad, copiar, descompactar, eliminar y restablecer versión (en inglés, *restore*).

Un actualizador automático consta de dos partes esenciales; la primera, una herramienta desarrollada para el programa de actualización que permita a los usuarios crear y publicar fácilmente los archivos que conforman el paquete actualizador, y como segunda, un programa de actualización tan flexible que sea capaz de interpretar las tareas especificadas en el paquete actualizador y mantener al tanto al usuario a través de mensajes sobre el estado en que se encuentra la actualización.

El funcionamiento de las aplicaciones genera inconsistencias, acumulación de datos, pérdidas de rendimiento y dependencias que pueden ser corregidas por una actualización evitando posibles fallas.

El objetivo de los actualizadores automáticos es ayudar a las empresas a mejorar su operativa y colaborar con ellas para que sus sistemas de software mantengan un funcionamiento libre de problemas que garanticen la seguridad global, pues una vulnerabilidad en un programa constituye una puerta para posibles ataques de programas malignos(en inglés, *malwares)* que operan generalmente sobre errores. Además, establece costes de mantenimiento reducidos al tiempo que proporciona actualizaciones del producto y servicios de planificación para llevar a cabo las mismas desde nuestras propias instalaciones.

Como se ha descrito, es necesario realizar actualizaciones tecnológicas y de integración con diferentes interfaces, y añadir nuevas funcionalidades para mejorar el rendimiento y la usabilidad de las aplicaciones

informáticas. Actualmente en el Centro de Salud de la Facultad 7 no se cuenta con una herramienta que cubra estas necesidades y garantice la inexistencia de los problemas antes enunciados. En el mejor de los casos se disponen de módulos básicos que sólo incorporan las acciones fundamentales de descarga, copia, ejecución y eliminación. Estos módulos no constituyen un software estándar para todas las soluciones desarrolladas en el centro.

La ausencia de un mecanismo que facilite la mejora continua de las soluciones alas constituye una gran limitante para brindar un servicio de atención y soporte óptimo.

Por lo antes expuestos, se define el siguiente **problema a resolver:** Carencia de una solución informática que permita la gestión de las actualizaciones automáticas para las aplicaciones desarrolladas en el Centro de Informática Médica.

Como **objeto de estudio** se propone el proceso de gestión de la información de las aplicaciones informáticas desarrolladas en el Centro de Informática Médica, de manera que el **campo de acción** se enmarca en el proceso de gestión de la información de las actualizaciones automáticas para las soluciones elaboradas en el Centro de Informática Médica.

Basado en esa idea se identifica como **objetivo general** del presente trabajo: Desarrollar un Actualizador Automático para soluciones alas, que proporcione un servicio flexible de soporte y mantenimiento a dichas soluciones.

Para dar cumplimiento al objetivo anteriormente planteado se definen las siguientes **tareas a desarrollar:**

1. Analizar los procesos asociados a la actualización automática de los sistemas informáticos.
2. Evaluar las tendencias actuales en el mundo de los actualizadores automáticos.
3. Asimilar la arquitectura definida por el Departamento de Sistemas de Apoyo a la Salud para el desarrollo de sus aplicaciones.
4. Obtener mediante el Proceso Unificado de Desarrollo, los flujos de trabajo “Modelado de Negocio”, “Requerimientos”, “Análisis y Diseño” e “Implementación”.
5. Implementar los procesos de negocio relacionados con las tareas de descarga, copias de seguridad, copiar, ejecución, eliminación y *restore*, propias de un actualizador automático.

Con la puesta en marcha de la solución obtenida en esta investigación se logran alcanzar los siguientes beneficios:

- ✓ Garantía de la estabilidad en las soluciones desplegadas por el Centro de Informática Médica.
- ✓ Continuidad del soporte y mantenimiento a las soluciones desplegadas de manera progresiva.
- ✓ Disminución de gastos de recursos por concepto de mantenimiento de las soluciones.
- ✓ Aumento de la velocidad de respuestas a problemas o incidencias presentados.
- ✓ Eliminación de la dependencias de servicios de terceros para brindarle mantenimiento a los clientes.

Con el propósito de organizar y darle una estructura al trabajo se ha decidido dividir el mismo en cuatro capítulos:

Capítulo 1: Fundamentación teórica: Estudio preliminar de los actualizadores automáticos existentes en Cuba y el mundo. Tecnologías, metodologías y herramientas de desarrollo a utilizar.

Capítulo 2: Características del sistema: Descripción de las funcionalidades de la solución propuesta a la problemática planteada.

Capítulo 3: Análisis y Diseño del sistema: Modelación y construcción de la estructura de la aplicación.

Capítulo 4: Implementación: Implementación de las clases y subsistemas en términos de componentes de la solución propuesta.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

Existen conceptos asociados a los procesos de gestión de actualizaciones automáticas que pueden resultar desconocidos y son de vital importancia para la comprensión de la investigación. Ayuda en este tema, el conocimiento del estado actual de estos, el análisis y comparación de experiencias precedentes, tendencias, metodologías, tecnologías y herramientas actuales que fundamentan la selección que se presenta en capítulos posteriores.

1.1 Conceptos básicos relacionados con el dominio del problema

Para comprender los términos empleados en esta investigación se ha de conocer el ámbito de los actualizadores automáticos, por lo que a continuación se explican algunos conceptos que no forman parte del lenguaje común.

Aplicaciones informáticas: Es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajo. Esto lo diferencia principalmente de otros tipos de programas como los sistemas operativos (que hacen funcionar al ordenador), las utilidades (que realizan tareas de mantenimiento o de uso general) y los lenguajes de programación (con los cuales se crean los programas informáticos). Estas aplicaciones informáticas son objeto de actualizaciones.

Se designa con el término **actualización informática** a aquella tarea o actividad que supone la puesta al día y/o sustitución de una información contenida en un registro o archivo por otra más reciente. [2]

Automático es aquello perteneciente o relativo al autómata. Este término proviene del griego *automatos* que significa “con movimiento propio” o “espontáneo”. Por lo tanto, la noción de automático puede hacer referencia a distintas cuestiones. [3] Una de ellas es la carencia de supervisión de terceros que pueden ser personas o sistemas en un proceso definido. Un mecanismo automático funciona por sí solo, ya sea en su totalidad o en una parte.

Una vez enunciados los dos conceptos anteriores se define como **actualización automática** a la función que realiza un **actualizador automático** (*gestor de actualizaciones*) para comprobar si se ha publicado una nueva actualización del software instalado, para ello se exploran los repositorios de software en busca

de nuevas versiones. Estas, normalmente, contienen reparaciones de errores e incluyen nuevas capacidades y cualidades a nuestras aplicaciones, pero también pueden contener actualizaciones de seguridad. Es por todo esto que se recomienda utilizar el gestor de actualizaciones de forma periódica para asegurarse de que el sistema está al día y tan seguro como sea posible.

1.2 Antecedentes

Cualquier esfuerzo de Ingeniería del Software – si termina con éxito – acaba por producir un determinado producto software, orientado a satisfacer ciertos requisitos previamente establecidos. El mantenimiento en este contexto se entiende de manera general como las actividades de cambio de ese producto. Ahora bien, el cambio se puede entender de diferentes maneras. Comenzando por lo básico, la definición de "Mantenimiento del Software" del estándar IEEE 1219 es: "El mantenimiento del software es la modificación de un producto software después de la entrega para corregir fallos, para mejorar el rendimiento u otros atributos, o para adaptar el producto a un entorno modificado". [4] el mismo fue publicado el 25 de junio de 1998, hasta esa fecha único estándar que íntegramente se ocupa del proceso de Mantenimiento del Software.

El estándar (IEEE 1219) define cambios en un producto software a través de un proceso de mantenimiento dividido en fases. Este proceso es iterativo y en cascada, con una gran semejanza al ciclo de vida del desarrollo clásico. En él se detalla un proceso iterativo para gestionar y realizar las actividades de mantenimiento. Esta definición implica que las actividades de mantenimiento de un producto comienzan en el tiempo sólo después de que el producto se ha entregado, es decir, después de que el producto está en operación, sin embargo, algunos autores como Pigoski (1997), resaltan la necesidad de comenzar a considerar el mantenimiento desde el mismo momento en que comienza el desarrollo.

De la definición de mantenimiento del estándar IEEE 1219 cabe distinguir tres causas fundamentales que desencadenan las actividades de mantenimiento:

1. Eliminación de defectos del producto software.

2. Adaptar el producto software a cambios en el entorno.

3. Incluir mejoras en el diseño.

Estas causas son resultado de tener que modificar el software para que cumpla con los requisitos del usuario ya establecidos (caso 1), para que siga cumpliéndolos cuando cambia su entorno (caso 2), o cuando se quiere mejorar la manera en que los cumple (caso 3). Por otro lado, la definición anterior implica que el mantenimiento debido a los defectos es a posteriori, es decir, se desencadena cuando el defecto tiene como resultado un fallo que se detecta.

En ocasiones, se realizan actividades de mantenimiento preventivo, que intentan detectar y corregir fallos latentes (que se supone pueden existir, aunque aún no se han “manifestado”). Estas causas tienen su correlación directa con las denominadas “categorías de mantenimiento” definidas por Lientz y Swanson 2(1978) en el estándar ISO/IEC 14764:

1. Mantenimiento correctivo: modificaciones reactivas a un producto software hechas después de la entrega para corregir defectos descubiertos.

2. Mantenimiento adaptativo: modificación de un producto software realizada después de la entrega para permitir que el mismo siga pudiéndose utilizar en un entorno diferente.

3. Mantenimiento perfectivo: modificación de un producto software después de la entrega para mejorar el rendimiento o la mantenibilidad.

Como consecuencia y necesidad de dar soporte y mantenimiento a los sistemas es que surgen las actualizaciones. La empresa multinacional estadounidense *Microsoft Corporation*, fundada en 1975 por Bill Gates y Paul Allen, fue de las primeras compañías en apostar por la idea de hacer de las actualizaciones, un proceso sencillo y en alguna medida automatizado. El sistema operativo gráfico Windows 98 publicado el 25 de junio de 1998 fue el encargado de poner en marcha este criterio. El mismo incluyó el módulo Windows Update (módulo vía red de actualización de Windows), que contactaba al sitio oficial con un ActiveX que permitía ver la información del sistema y descargar las actualizaciones adecuadas. Con el lanzamiento de Windows Millennium Edition en 2000, Microsoft presentó las actualizaciones automáticas permitiendo descargar e instalar actualizaciones de forma oculta como un reemplazo de la herramienta Notificación de actualización crítica; con el objetivo de hacer más fácil la búsqueda de actualizaciones.

Actualmente Microsoft dispone de un servicio que ayuda a los usuarios a mantener al día la seguridad y programas en sus computadoras: Microsoft Update. Al visitar su sitio web, la herramienta analiza el sistema para comprobar lo que el equipo necesita y ajusta sus recomendaciones al software de Microsoft instalado en la computadora del usuario. Estas recomendaciones pueden ser actualizaciones de alta prioridad o actualizaciones opcionales.

El sistema operativo GNU/Linux por su parte, se basó en la teoría de que es recomendable configurar las aplicaciones instaladas para que sólo se puedan actualizar de modo manual y cuando el usuario lo indique. Este problema está resuelto con la utilización de Synaptic Package Manager, un programa para la gestión de paquetes de aplicaciones. Funciona en distribuciones como Debian y Ubuntu. Utiliza los diferentes repositorios de archivos para la gestión y actualización de todos los paquetes instalados en el sistema y otros más que estuvieran disponibles para su puesta en marcha.

La última versión del sistema operativo Mac OS X "Leopard" contiene un actualizador de software que avisa de nuevas actualizaciones del sistema operativo y de las aplicaciones desarrolladas y controladas por Apple, como iLife, iWork o iTunes. Sin embargo, esta herramienta no gestiona las aplicaciones de terceros. Una solución es Application Update (acceso directo), un *widget* gratuito que busca todas las actualizaciones disponibles entre las aplicaciones instaladas en el ordenador. Para ello, utiliza la base de datos de la página de aplicaciones Versiontracker.com y compara las nuevas versiones con las detectadas dentro de la carpeta de aplicaciones en Mac Os X. [5]

1.3 Sistemas automatizados existentes vinculados al campo de acción

Los sistemas operativos disponen de un gestor de actualizaciones que resulta muy útil para mantener sus registros y operativa al día, pero que deja fuera todas las aplicaciones de terceros instaladas en el ordenador. Debido a la seguridad y flexibilidad que ofrecieron los actualizadores automáticos presentados por la Microsoft y GNU/Linux, tuvieron gran aceptación y a su vez despertaron el interés de otras compañías desarrolladoras de las más diversas aplicaciones, que como sistemas informáticos presentaban las mismas necesidades de actualización que los sistemas operativos.

La solución ha sido la creación de varios gestores de actualizaciones que no sólo se limitan a actualizar diferentes programas externos sino que también los instalan. Otros por su parte, se especializan en

actualizar soluciones informáticas de interés específico desarrollándolos un poco más dirigido y enfocado a las necesidades de sus aplicaciones y a la demanda de sus usuarios con el objetivo de alcanzar niveles máximos de usabilidad, rendimiento y utilidad.

En el competitivo mercado del software existen varios ejemplos de actualizadores automáticos. AppSnap, SUMo, UpdateStar, AppUpdater, GetIt son algunos de los más usados, dentro de los que gestionan un variado número de aplicaciones que no guardan afinidad alguna. Mientras que el actualizador SIGHO se limita a brindar servicios de actualización a una única entidad de software. El mismo está dirigido al personal técnico encargado del mantenimiento del Sistema de Información para la Gerencia Hospitalaria (SIGHO), un software desarrollado en México. De todos ellos, sólo se analizan los más utilizados y que resultan de interés para la investigación por sus características.

1.3.1 Appsnap

AppSnap es una herramienta sencilla y fácil de utilizar pero tiene el inconveniente que dispone de un conjunto limitado de aplicaciones soportadas: su base de datos no es muy extensa ya que cuenta con 400 entradas, aunque se pueden sugerir nuevas aplicaciones. Entre las características principales de AppSnap se encuentran: [6]

1. Detecta automáticamente la última versión disponible del software instalado.
2. Descarga, instala, actualiza y desinstala las aplicaciones compatibles.
3. Base de datos de aplicaciones centralizada y actualizada a diario.
4. Crear un repositorio único de solicitud para ser utilizado por AppSnap en una intranet.
5. Soporte para configuraciones proxy.
6. Descargas paralelas con información de progreso.
7. Filtros de aplicaciones por categorías y palabras claves.

AppSnap puede resultar útil en el despliegue de aplicaciones a través de varias máquinas en una intranet. Para ello establece dos configuraciones posibles las cuales se han clasificado en dos tipos a conveniencia (online u offline) para un mejor entendimiento de los lectores.

La clasificación online se refiere a cuando se tiene una estación conectada directamente a Internet donde con una simple configuración del archivo *config.ini* que se genera en la carpeta de instalación del AppSnap se puede especificar el directorio donde serán instaladas las aplicaciones por defecto, además del usuario y contraseña necesarios para autenticar contra el proxy. Esta configuración brinda la posibilidad de hacer aún más trasparente el proceso.

La clasificación offline se refiere a cuando tenemos uno o varios grupos de estaciones sin conexión a Internet, y un servidor central con AppSnap instalado y acceso a Internet.

¿Cuál es su principio de funcionamiento? Dicho servidor central de aplicaciones descargará toda la información de la versión necesaria y los instaladores hacia la carpeta caché de AppSnap y de esta manera mantendrá actualizada su base de datos de aplicaciones. Para ello puede utilizar la configuración anterior del fichero *config.ini*. Esta carpeta caché es compartida por la red al resto de las estaciones para que las mismas puedan conectarse y descargar los recursos disponibles a través de instancias remotas de AppSnap. Con este mecanismo las estaciones sin acceso a Internet pueden también mantener actualizada su base datos de aplicaciones.

Como se aprecia, en ambas configuraciones (online u offline) es necesaria la conexión a Internet para realizar la descarga de la base de datos centralizada disponible en el portal oficial de AppSnap; donde estarán un conjunto de aplicaciones que no son de interés, sin posibilidad alguna de creación de un servicio con repositorio y base de datos propia que contenga solo las aplicaciones de interés para el negocio. Esto evidencia la existencia obligatoria de un servicio externo a nuestro negocio y que por lo tanto está fuera de control. Por otro lado, el software descargado sólo está disponible en los idiomas búlgaro, danés, holandés, inglés, francés, alemán y ruso, lo que podría ser inservible a usuarios que no dominen ninguna de estas lenguas.

La mayor cualidad que presenta AppSnap es que es gratis y de código abierto, constituyendo dos elementos importantes a la hora de evaluar las aplicaciones que podrían dar solución al problema, pero no representa un factor determinante.

1.3.2 Appupdater

Appupdater es otra de las herramientas de instalación y actualización de programas más utilizada en el mundo que se maneja en línea de comandos. Prácticamente no hay diferencia con los comandos apt-get o yum de Linux, que son la vía tradicional de instalación de aplicaciones en este sistema operativo. Appupdater actualiza periódicamente la base de datos con información sobre cientos de aplicaciones para que el usuario la consulte e instale lo que desee en línea y a petición. Además de ser multiplataforma, incluye soporte para dispositivos USB y se pueden sugerir nuevas aplicaciones que se incluirán en la lista de aplicaciones soportadas. Crear un repositorio personalizado si se desea es otra de las posibilidades que brinda Appupdater. A continuación se muestra un listado de reglas para la creación de un repositorio propio.

1. Sólo programas en inglés para Windows.
2. Sólo software estables no betas.
3. El software debe estar libremente publicado sin contraseñas, registros ni formularios web.
4. No plugins a menos que tengan su propio instalador.
5. No drivers.

Las nuevas aplicaciones serán agregadas a su solicitud, si cumple los criterios anteriores. [7]

Aparentemente Appupdater es la solución perfecta para actualizar todos los desarrollos independientes una vez que se cumplan con las reglas de creación de repositorio. Nótese pues que la primera regla constituye una barrera para los desarrolladores de soluciones informáticas en español: que se ven imposibilitados de crear un repositorio propio y como consecuencia no poder utilizar Appupdater. Entre las principales características de Appupdater destacan algunos aspectos que nos hacen desistir de la idea de utilizarlo como por ejemplo:

- ✓ Appupdater descarga una lista de todas las aplicaciones, lo que instala no es enviado ni mostrado, por lo que no se tiene conocimiento de los ficheros o registros actualizados.
- ✓ Durante la instalación del programa se descarga directamente desde el sitio web de descarga, no es posible hacer un seguimiento de lo que se está instalando y si falla la conexión ocurre un error.
- ✓ Apppdater sólo envía la información que realiza un navegador web normal.

- ✓ Problemas con versiones en español de programas en Windows u otras versiones.
- ✓ No se pueden instalar pluggins por separado a no ser que exista un instalador para cada uno. En ese caso se adopta la convención de que cada pluggins es como otro programa a instalar y chequear por actualizaciones.
- ✓ No permite chequear que el programa a actualizar necesite instalar un parche de Windows y este ya exista, por lo que esto produciría un error.

1.3.3 Actualizador SIGHO

Con el objetivo de mantener actualizadas las aplicaciones, agregando funcionalidad y mejorando calidad, se continúa con los esfuerzos de obtener nuevas versiones que se puedan instalar en las unidades médicas donde se encuentre funcionando el SIGHO. El actualizador SIGHO ayuda a mantener al día las versiones de manera más organizada apoyando al personal técnico encargado. Para ejecutar el actualizador del SIGHO es importante cubrir los siguientes pre-requisitos para el correcto funcionamiento de la aplicación.

Es necesario que en el equipo donde se ejecutará la aplicación esté previamente instalado el SIGHO, ya que el actualizador no podrá ejecutarse en el servidor de otra manera. El personal encargado no podrá realizar ninguna operación, sin antes tener una cuenta de usuario del sistema SIGHO.

Ventajas de utilizar el actualizador automático SIGHO.

- ✓ Se evitan cargas de trabajo.
- ✓ Reemplaza ejecutables automáticamente.
- ✓ Registra y des-registra controles automáticamente.
- ✓ Compara versiones de controles y ejecutables para que se utilicen las más actuales.

Desventajas de no utilizar el actualizador automático SIGHO.

- ✓ El sistema puede generar errores por no contar con los controles correctos.
- ✓ Realizar manualmente la sustitución de ejecutables.
- ✓ Realizar manualmente el registro y des-registro de los controles.

- ✓ No se cuenta con la seguridad de que todos los equipos cuenten con las versiones más actuales.

Este actualizador automático tiene como finalidad reducir cargas de trabajo, sustituyendo automáticamente ejecutables y controles de versiones actualizadas, sin la necesidad de que esto se realice manualmente. Entre las características que más destacan de esta herramienta se encuentran:

1. Cuenta con un sistema de autenticación para evitar que personal ajeno a la institución en que se utiliza, o sin los permisos necesarios, acceda a la aplicación.
2. Al iniciar el proceso de actualización muestra una pantalla de Atención, en la que se le indicará que realice un respaldo de todo el sistema, es decir, base de datos, controles, ejecutables, reportes y plantillas para si existe algún error durante el proceso de actualización poder establecer la configuración anterior y evitar la pérdida de información.
3. Posee una pestaña llamada Historial de Versiones que visualiza un listado de todas las versiones que se han instalado en su equipo, así como la fecha y el personal que ejecutó el proceso de actualización.

Como se observa, la utilización de esta herramienta está condicionada por la utilización del Sistema de Información para la Gerencia Hospitalaria SIGHO (su instalación es paralela al mismo) pero contempla una serie de opciones y características que son de interés para la solución propuesta más adelante.

Luego de haber analizado cada una de las características presentes en los software mencionados anteriormente, que forman parte de los gestores de búsqueda de actualizaciones más usados por compañías desarrolladoras de software independientes, se han identificado un conjunto de aspectos que hasta cierto punto pueden ser debilidades, excesos o carencias que presentan estos sistemas para el dominio del negocio.

- ✓ Debido a que estos sistemas pueden actualizar un conjunto de aplicaciones al mismo tiempo, levantan un gran número de procesos paralelos e independientes que tienen múltiples fuentes de información operando; como son los repositorios. Esto puede provocar un desbordamiento de la cola de procesos si no son bien implementados y monitoreados.
- ✓ No permiten la creación de un repositorio propio donde sólo se encuentren las aplicaciones que sean de interés para las empresas e instituciones desarrolladoras. Sólo Appupdater brinda esta

posibilidad, pero para ello hay que cumplir con un conjunto de reglas de repositorio y luego contactarlos para su inclusión.

- ✓ Entre los propósitos de un actualizador automático está solucionar problemas de seguridad a las aplicaciones que actualizan, pero ellos por si solos no garantizan su seguridad. Tanto la conexión a los repositorios compartidos por un recurso local como el uso propio de los actualizadores en la mayoría de los casos, no requieren de autenticación. Esto hace que los procesos de actualización no sean controlados y evita en ocasiones tener que generar reportes de interés, como por ejemplo un historial de versiones donde se relacione el personal que ejecutó el proceso en distintos momentos.
- ✓ Una actualización es concebida por estos actualizadores como un nuevo *release* del software, cuando puede ser tan sencilla como el remplazo de uno de los componentes del software.
- ✓ Estos actualizadores llevan un control interno de los componentes que están siendo actualizados, pero a su vez los usuarios no están siendo notificados. No se envía un mensaje de notificación hasta que culmina el proceso de manera factible o el producto de un error. De tal manera, ante la ocurrencia de un problema, el usuario no puede saber en qué nivel de la actualización se produjo el problema.
- ✓ La mayoría de los actualizadores automáticos disponibles son en inglés, lo que presupone una limitante para posibles usuarios de habla hispana.
- ✓ Los manuales dirigidos a orientar a los usuarios en cuanto a requerimientos, funcionalidades, escenarios comunes, aspectos de configuración y uso, son mínimos.
- ✓ Las interfaces de usuarios son poco descriptivas en correspondencia con las funcionalidades que implementan.

1.4 Tendencias, tecnologías y herramientas

1.4.1 Estilos arquitectónicos y patrones

Un **estilo arquitectónico** define las reglas generales de organización en términos de un patrón y las restricciones en la forma y la estructura de un grupo numeroso y variado de sistemas de software.

Un **patrón** es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. Cada patrón describe

un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema. Los patrones de diseño se han convertido en la metodología por excelencia de diseño y ulterior programación de una solución importante.

Los patrones de diseño pretenden:

1. Proporcionar catálogos de elementos reusables en el diseño de sistemas.
2. Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
3. Formalizar un vocabulario común entre diseñadores.
4. Estandarizar el modo en que se realiza el diseño.
5. Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Para el desarrollo del Actualizador Automático se propone la utilización de los estilos en capas y orientado a objetos de la familia de estilos arquitectónicos “Llamada y Retorno”.

1.4.2 Arquitectura en tres capas

El **estilo arquitectónico en capas** es un estilo de programación cuyo objetivo primordial es la separación y agrupamiento de los componentes del software atendiendo a la función que cumplen en el mismo. Para realizar el agrupamiento se tiene en cuenta las funcionalidades relacionadas con el usuario del sistema, así como la información que este maneja y las operaciones que realiza sobre la misma en dependencia de la complejidad que se necesita que tenga el sistema. Esta división muchas veces se hace en tres capas: la capa de presentación, capa de negocio y la capa de datos, aunque para la solución propuesta se sugiere remplazar la capa de datos; y en su lugar ubicar la capa de tareas que será la encargada de ejecutar y controlar las acciones propias para cada actualización.

Capa de presentación o interfaz de usuario: presenta el sistema al usuario. Captura y comunica la información del usuario dando un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.[8]

Capa de negocio o procesos de actualización: es donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) pues es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de tareas, para solicitar el inicio de una nueva tarea y ser notificado al término de la misma.

Capa de tareas: es la encargada de manipular la solicitud de inicio de las tareas y notificar a la capa superior sobre el estado de las mismas (progreso o terminada) en cada momento para que esta a su vez presente al usuario a través de la capa de presentación el progreso de la actualización. Su funcionamiento básico está descrito por el trabajo con ficheros XML haciendo uso de Nant, una herramienta libre y de código abierto para la automatización de procesos. Para lograr una integración entre Nant y C# se hace uso del esquema definido por Nant a partir de cual se genera las clase que lo representa que proporciona un acceso fácil a secuencias XML validadas por este esquema.

Las ventajas del estilo en capas son obvias. Primeramente, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales. En segundo lugar, el estilo admite muy naturalmente optimizaciones y refinamientos. En tercer lugar, proporciona una amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes. Esto conduce a la posibilidad de definir interfaces de capa estándar, a partir de las cuales se pueden construir extensiones o prestaciones específicas.[9]

1.4.3 Arquitectura Orientada a Objeto

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, [10] los objetos representan una clase de componentes que ellos llaman *managers*, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. En la semblanza de estos autores no se establece como cuestión

definitoria el principio de herencia. Ellos piensan que, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objetos, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Principales características de las arquitecturas Orientada a Objeto (OO):

- ✓ Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.
- ✓ En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos. Hay muchas variantes del estilo; algunos sistemas, por ejemplo, admiten que los objetos sean tareas concurrentes; otros permiten que los objetos posean múltiples interfaces.

1.4.4 Patrones de Asignación de Responsabilidades (GRASP)

Un sistema orientado a objetos se compone de objetos que envían mensajes a otros objetos para que lleven a cabo las operaciones requeridas. Los diagramas de interacción describen gráficamente estas operaciones, a partir de los objetos en interacción, que se responsabilizan de una actividad determinada.

La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos. En los patrones GRASP se codifican algunos de los principios, que se aplican al preparar los diagramas de interacción.

Los diseñadores expertos en orientación a objetos (y también otros diseñadores de software) van formando un amplio repertorio de principios generales y de expresiones que los guían al crear software. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas.

Los patrones generales de software para asignar responsabilidades GRASP (en inglés General Responsibility Asignment Software Patterns) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

El nombre se eligió para indicar la importancia de captar (*grasping* en inglés) estos principios, si se quiere diseñar eficazmente el software orientado a objetos. A continuación se describen los patrones básicos de asignación de responsabilidades. [11]

Patrón Experto:

Problema:

¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos? Un modelo de clase puede definir docenas y hasta cientos de clases de software, y una aplicación tal vez requiera el cumplimiento de cientos o miles de responsabilidades. Si estas se asignan en forma adecuada, los sistemas tienden a ser más fáciles de entender, mantener y ampliar, y se nos presenta la oportunidad de reutilizar los componentes en futuras aplicaciones.

Solución:

Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Patrón Creador:

Problema:

¿Quién debería ser responsable de crear una nueva instancia de alguna clase? La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

Solución:

Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- ✓ B agrega los objetos A.
- ✓ B contiene los objetos A.
- ✓ B registra las instancias de los objetos A o
- ✓ B utiliza especialmente los objetos A.
- ✓ B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A). B es un creador de los objetos A.

Si existe más de una opción, es conveniente elegir la clase B que agregue o contenga la clase A.

Patrón Bajo Acoplamiento:

Problema:

¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.

Solución:

Asignar una responsabilidad para mantener bajo acoplamiento. El grado de acoplamiento no puede considerarse aisladamente de otros principios como Experto y Alta Cohesión. Sin embargo, es un factor a considerar cuando se intente mejorar el diseño.

Patrón Alta Cohesión:

Problema:

¿Cómo mantener la complejidad dentro de límites manejables? La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo.

Solución:

Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Patrón Controlador:

Problema:

¿Quién debería encargarse de atender un evento del sistema? Un evento del sistema es un evento de alto nivel generado por un actor externo; es un evento de entrada externa. Se asocia a operaciones del sistema: las que emite en respuesta a los eventos del sistema. Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema. Define además el método de su operación.

Solución:

Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

1.4.5 Tecnologías de desarrollo y herramientas

- **Enterprise Architect 7.1**

Enterprise Architect es una herramienta CASE (Computer Aided Software Engineering) para el diseño y construcción de sistemas de software. EA soporta la especificación de UML 2.0, que describe un lenguaje visual por el cual se pueden definir mapas o modelos de un proyecto.

EA es una herramienta progresiva que cubre todos los aspectos del ciclo de desarrollo, proporcionando una trazabilidad completa desde la fase inicial del diseño a través del despliegue y mantenimiento. También provee soporte para pruebas, mantenimiento y control de cambio.

Algunas de las características claves de Enterprise Architect son:

- ✓ Crear elementos del modelo UML para un amplio alcance de objetivos.
- ✓ Ubicar esos elementos en diagramas y paquetes.
- ✓ Crear conectores entre elementos.
- ✓ Documentar los elementos que ha creado.
- ✓ Generar código para el software que está construyendo.
- ✓ Realizar ingeniería reversa del código existente en varios lenguajes.
- ✓ Control de versiones con cualquier herramienta SCC

Enterprise Architect sustenta todos los diagramas y modelos UML. Puede modelar procesos de negocio, sitios web, interfaces de usuario, redes, configuraciones de hardware, mensajes y más. Estimar el tamaño de su proyecto en esfuerzo de trabajo en horas. Capturar y trazar requisitos, recursos, planes de prueba, solicitudes de cambio y defectos. Desde los conceptos iníciales hasta el mantenimiento y soporte, Enterprise Architect tiene las características que precisa para diseñar y administrar su desarrollo e implementación. [12]

1.4.6 Visual Studio 2005

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones Web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones Web ASP y Servicios Web XML .[13] Visual Studio 2005 por su parte provee una serie de herramientas para desarrollo, así como características de debugging, funcionalidad en base de datos y características innovadoras para la creación de aplicaciones en una variedad de plataformas. Incluye realces como un diseñador visual para desarrollo rápido con el .NET Framework 3.5. Además Visual Studio 2005 provee a desarrolladores con todas las herramientas y el Framework el poder crear aplicaciones web con el soporte de AJAX. También permite escoger entre múltiples versiones del Framework con el mismo entorno de desarrollo, así nosotros podemos desarrollar en la versión que queramos ya sea en .NET Framework 2.0, 3.0 o 3.5, entiendo así que soporta un gran variedad de proyectos en la versión X en el mismo entorno de desarrollo.

1.4.7 NAnt

NAnt es una herramienta de código abierto para automatizar procesos. Constituye una adaptación de Apache Ant para **.**NET**.** Se puede extender mediante clases *Tasks (*tareas*)*, no mediante comandos *Shell*, como la mayoría de estas herramientas. Esta característica la hace multiplataforma. Los archivos de configuración se basan en sintaxis XML, tienen extensión *.build* y están compuestos por cuatro tipos de ítems*(etiquetas): Tasks(tareas)*, *Targets(nodos)*, *Properties(propiedades)* y *Projects(proyectos)*, en los nodos *Target* se ejecutan una o más *tasks*, si es necesario, estableciendo un orden de precedencia y también dependencias. NAnt también puede ser ejecutado desde **.**NET en eso se basa su compatibilidad con .NET. Entre las tareas más importantes que implementa la más reciente versión de NAnt se encuentran:

1. <Copy> Copia un archivo o conjunto de archivos a un nuevo archivo o directorio.
2. <Delete> Elimina o un solo archivo, todos los archivos en un directorio especificado y sus subdirectorios, o un conjunto de archivos especificados por uno o más conjuntos de archivos.
3. <Gunzip> Expande un archivo comprimido mediante la compresión GZip.

4. <Move> Mueve un archivo o conjunto de archivos a un nuevo archivo o directorio.
5. <Zip> Crea un archivo zip de los conjuntos de archivos especificado.
6. <Tar> Crea un archivo tar de los conjuntos de archivos especificado.
7. <Readregistry> Lee un valor o conjunto de valores del Registro de Windows en una o más propiedades de NAnt.

1.4.8 XML

Lenguaje de marcas ampliable (siglas en inglés de Extensible Markup Language). Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Permite definir la gramática de lenguajes específico, por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML. [15] XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil. El fichero de compilación *.build* en el que se especifican las tareas a desarrollar por NAnt está escrito íntegramente en lenguaje XML.

1.4.9 Modelo de desarrollo del software (CMMI)

A lo largo de todo el ciclo de vida del desarrollo del software se utiliza el modelo CMMI (Modelo Integrado de Madurez de Capacidades), definido por el proyecto SAS, para la evaluación de los procesos que se realizan para producir software. CMMI es una guía para mejorar procesos y comprobar la capacidad de un grupo al ejecutarlos, un modelo de madurez, un marco para diagnosticar el estado de la mejora que indica que debe hacer los procesos, y tiene cinco niveles de madurez. Este contiene dos representaciones la continua y la escalonada, la utilizada para este trabajo es la escalonada y el nivel 2 de madurez de CMMI, esta representación ofrece una manera estructurada y sistemática de enfrentar la mejora de proceso, ejecutando un paso cada vez. El nivel 2 de madurez de CMMI se define en siete áreas de control que son

REQM (Gestión de Requerimientos), PP (Panificación de Proyecto), PMC (Seguimiento y Control de Proyectos), SAM (Gestión de Acuerdos con Proveedores), MA (Medición y Análisis), PPQA (Aseguramiento de la Calidad de Procesos y Productos), CM (Gestión de la Configuración). El área de control que se utilizara es REQM, el objetivo de esta función es administrar los requisitos de los productos y componentes de productos del proyecto. Identificar inconsistencias entre esos requisitos y los planes y productos de trabajo del proyecto.

Como metodología de desarrollo se utiliza Proceso Unificado de Desarrollo (RUP). Es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto. Junto con el Lenguaje Unificado de Modelado UML 2.0 constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

1.4.10 Lenguaje Unificado de Modelado (UML 2.0)

UML es un lenguaje de modelado visual que comenzó a desarrollarse a partir de 1994. Permite visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. Permite la modelación de sistemas con tecnología orientada a objetos. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como RUP), pero no especifica en sí mismo qué metodología o proceso utilizar. Este lenguaje de modelado formal permite tener un mayor rigor en la especificación, realizar una verificación y validación del modelo desarrollado, automatizar determinados procesos y generar código a partir de los modelos y a la inversa. Esto último permite que el modelo y el código estén actualizados. [17]

Ventajas de UML 2.0:

- ✓ Produce un aumento en la calidad del desarrollo.
- ✓ Reduce los costos del proyecto.
- ✓ Mejora en un 50% o más los tiempos totales de desarrollo.
- ✓ Permite especificar la estructura y el comportamiento del sistema y comunicarlo a todos los integrantes del proyecto.

1.4.11 Lenguaje de programación

- **C Sharp(C#)**

El lenguaje de programación C# fue creado por el danés Anders Hejlsberg quien diseñó también los lenguajes Turbo Pascal y Delphi. El C**#** (pronunciado en inglés “C sharp” o en español “C sostenido”) es un lenguaje de programación orientado a objetos. Con este nuevo lenguaje se quiso mejorar con respecto de los dos lenguajes anteriores de los que deriva el C, y el C++. Con el C# se pretendió que incorporase las ventajas o mejoras que tiene el lenguaje JAVA. Así se consiguió que tuviese las ventajas del C, del C++, pero además la productividad que posee el lenguaje JAVA y se le denominó C#. Algunas de las características del lenguaje de programación C# son: Su código se puede tratar íntegramente como un objeto. Su sintaxis es muy similar a la del JAVA. Es un lenguaje orientado a objetos y a componentes. Armoniza la productividad del Visual Basic con el poder y la flexibilidad del C++. Ahorramos tiempo en la programación ya que tiene una librería de clases muy completa y bien diseñada. [18]

Un lenguaje moderno es el que proporciona las últimas características y herramientas para el desarrollo de la industria escalable, confiable y robusto de aplicaciones estándar de modo que C# lo es de manera implícita ya que simplifica y moderniza a C++ en las áreas de clases, espacios de nombres (*namespaces*), sobrecarga de métodos y manejo de excepciones. Además soporta todas las características propias del paradigma de programación orientada a objetos: *encapsulación, herencia y polimorfismo*.

Las principales características que hacen del lenguaje C# una elección inteligente son:

Sencillez e independencia de tipos: El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad de código.

Orientación a objetos: Como todo lenguaje de programación, C# es un lenguaje orientado a objetos, no admite ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de

definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

Orientación a componentes: La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente propiedades (similares a campos de acceso controlado), eventos (asociación controlada de funciones de respuesta a notificaciones) o atributos (información sobre un tipo o sus miembros).

Eficiente y con seguridad de tipos

Compatible: Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente código escrito en C# fragmento de código escritos es estos lenguajes, sino que el CLR también ofrece, a través de los llamados Plataform Invocation Services (Pinvoke).

Aunque C# forma parte de la plataforma .NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

Como resultado del estudio realizado en este capítulo, se puede concluir que los sistemas existentes que gestionan procesos asociados a las actualizaciones automáticas para aplicaciones no cumplen con los requisitos necesarios que demandan la actualización de soluciones alas. A pesar de ser multiplataforma y de código abierto no proveen un ambiente de configuración que permita adaptarlos al entorno de despliegue de las aplicaciones a actualizar. Por lo que se evidenció la necesidad de desarrollar un sistema que se ajuste más a las necesidades propias de las soluciones desarrolladas en el CESIM facilitando en gran medida el trabajo de actuales y futuros usuarios. La solución consiste en desarrollar una herramienta que requiera un gasto mínimo en recursos; para ello es de vital importancia reducir al máximo la necesidad de una conexión a Internet. Bastaría contar con un recurso compartido administrado por el personal interno de las instituciones, de bases de datos propias que sólo soporten las aplicaciones necesarias, con acceso desde todos los *host* miembros del dominio que dispongan de los permisos

necesarios para garantizar la integridad, disponibilidad y autenticidad de los recursos, y que además esté debidamente documentado para desarrollos futuros del mismo. De esta forma los usuarios tendrán una mayor accesibilidad a los servicios.

Se sugiere el empleo de las arquitectas, los patrones de diseño, modelos de desarrollo de software, tecnologías y herramientas descritas a lo largo de este capítulo para la obtención del sistema. Las mismas poseen un ambiente de integración propicio para el desarrollo de la herramienta propuesta. Las características y bondades que brindan, evidencian un desarrollo rápido y seguro, dejando espacio al análisis y empleo de buenas prácticas de diseño e implementación que garanticen un sistema eficiente.

CAPÍTULO 2. CARACTERÍSTICAS DEL SISTEMA

En este capítulo se describen los conceptos actuales que se relacionan con las tareas descargar, hacer copias de seguridad, copiar, eliminar y regresar al estado anterior conocida como Restore referentes a los actualizadores automáticos. En aras de tener una representación visual de esos conceptos u objetos se muestra el modelo conceptual, que es una representación visual estática del entorno real objeto del proyecto. Es decir, un diagrama con los objetos que existen (reales) relacionados con el proyecto que se desea acometer y las relaciones que hay entre ellos. Pero no son componentes (clases, etc.) de software (aunque algunos objetos del Modelo de Dominio pueden terminar siéndolo). Además, se definen los requerimientos funcionales y no funcionales, a partir de los cuales se representan los casos de uso del sistema, y se hace una especificación textual de los mismos.

2.1 Modelo Conceptual o Modelo de Dominio del sistema

El objetivo del Modelo de Dominio es ayudar a comprender los conceptos que utilizan los usuarios, los conceptos con los que trabajan y con los que deberá trabajar nuestra aplicación.

El proceso para su elaboración tiene tres pasos:

1. Identificar las Clases Conceptuales.
2. Dibujarlas en un Diagrama de Clases.
3. Añadir Relaciones y Atributos.

2.1.1 Identificar las Clases Conceptuales

Una "clase conceptual" es cualquier cosa que pertenezca al dominio. Por ejemplo, personas, máquinas, lugares, etc. Y también elementos intangibles como conceptos (venta, permiso, perfil, etc). Las clases suelen ser los sustantivos utilizados por los expertos al describir el dominio:

*"El **usuario** inicia la actualización de la **aplicación**."*

*"El **usuario** solicita el inicio de una nueva **tarea**."*

Técnico: Personal encargado de iniciar y asistir el flujo de eventos que comprende una actualización automática.

Aplicación: Herramientas, programas informáticos administradas por los técnicos, que serán objeto de las actualizaciones automáticas.

Reporte de Actualización: Reporte que se genera a solicitud de un técnico donde se especifican datos de las últimas actualizaciones efectuadas.

Paquete de Actualización: Contiene todos los componentes necesarios dígase (.Xml, .exe, .html, .cs entre muchos otros) para dar comienzo al flujo de actividades que engloba una actualización.

Manifiesto: La clase manifiesto es un documento XML que está contenido en el paquete de actualización. Posee un listado de información de todos los recursos y sus localizaciones en términos de tareas que se deben realizar en un orden establecido para llevar a cabo una actualización.

Repositorio: Lugar; recurso compartido donde se encuentra ubicado el paquete de actualización.

Tarea: Representan la unidad mínima de un manifiesto y describen una acción a realizar. Estas acciones puede ser copiar, eliminar, guardar, ejecutar, descompactar, mover entre otras.

Descarga: Acción que describe la vía por la cual se obtiene del repositorio el paquete de actualización y el mismo es almacenada en una dirección local de la PC.

Recursos de versión: Una clase recursos de versión no es más que aquella que tendrá el control de todas las versiones existentes, paralelo al desarrollo de una actualización con información única por cada una de las aplicaciones versionadas.

Actualización Automática: Flujo de actividades que describen el proceso llevado a cabo por un actualizador automático para completar el ciclo que comprende una actualización.

La lista anterior de los nombres de conceptos puede representarse gráficamente en la notación del diagrama de estructura estática de UML, a fin de mostrar la génesis del modelo conceptual.

2.1.2 Dibujar un Diagrama de Clases

El Modelo de Dominio queda recogido en un diagrama, representando cada clase por su nombre, dentro de un recuadro. Un error habitual es confundir las clases conceptuales con clases de software. Puede que finalmente, al realizar el diseño de clases, haya clases de software con el mismo nombre que las clases conceptuales del modelo de dominio. Pero en ningún caso son las mismas. Y aunque las diferencias puedan ser sutiles, si se aplican atributos y características propias del software a las clases conceptuales se estarán incluyendo restricciones innecesarias para el diseño de clases.

2.1.3 Añadir Relaciones y Atributos

Entre las clases existentes se establecen relaciones. Estas se identifican usualmente como los verbos utilizados por los expertos de dominio:

*"El usuario **inicia** la actualización de la aplicación."*

*"El usuario **solicita** el inicio de una nueva tarea."*

En el diagrama, cada relación queda representada por una línea que une las clases relacionadas (no sus atributos). Cada relación tiene también una multiplicidad (si es 1 a 1, n a 1, 0 ó 1...). Y esa no es tan fácil encontrarla. No suele figurar en las expresiones de los expertos. Para identificar la multiplicidad hay que tomar un papel activo y hacer preguntas concretas a los expertos, preguntas del tipo "¿Cuántas tareas puede incluir una actualización?" y proponerles situaciones aclaratorias ("Si un usuario solicita actualizar una aplicación y no se tiene acceso en ese momento al paquete de actualización ¿se guarda esa solicitud hasta que haya acceso?"). La multiplicidad de cada relación se representa sobre su línea, junto a la clase correspondiente.

Y por último, resta identificar los atributos de cada clase. En un modelo conceptual, es preferible que los atributos sean atributos simples o valores puros de datos. Entre los tipos comunes de atributos simples más frecuentes se cuentan: Booleano, Fecha, Número, Cadena (Texto) En una notación completa de cada uno de estos atributos podemos indicar: *visibilidad nombre: tipo = valor por defecto {propiedades}*

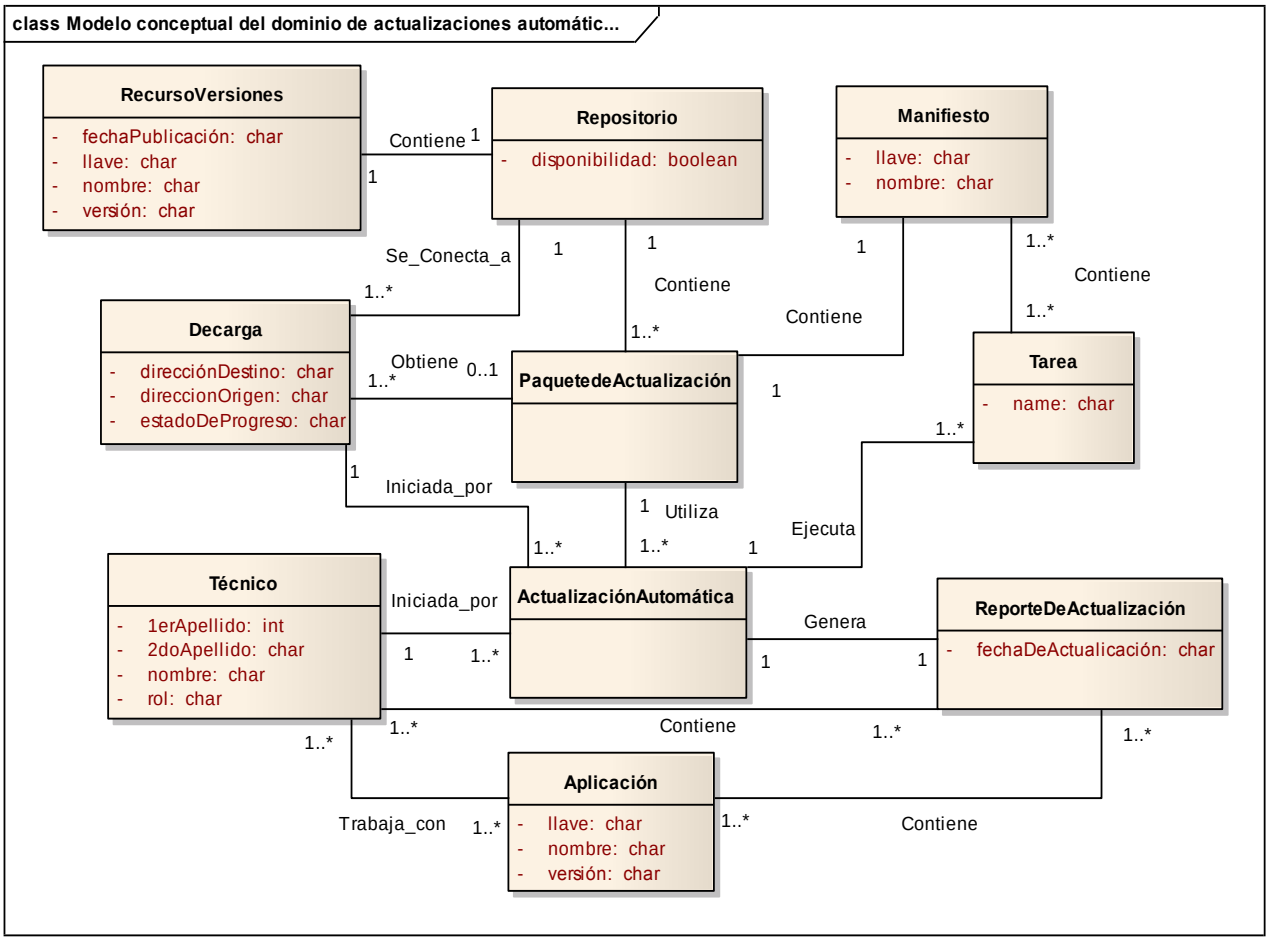

Figura 2.1 Modelo conceptual del dominio de actualizaciones automáticas con atributos.

Se ha creado un modelo relativamente útil al dominio de la aplicación al actualizador automático. No existe un modelo apropiado para todos los casos o circunstancias. Un buen modelo conceptual capta las abstracciones conceptuales esenciales y la información indispensable para comprender el dominio dentro del contexto de los requerimientos actuales; nos facilita además conocer el dominio: sus conceptos, su terminología y sus relaciones.

2.2 Propuesta del sistema

Teniendo en cuenta los estudios realizados y después de hacer un profundo análisis del objeto de estudio, se llega a la conclusión de implementar un sistema que gestione las actualización automática de las soluciones informáticas desarrolladas en el CESIM. El mismo, debe estar centrado en corregir errores que

son generados con el funcionamiento diario de las aplicaciones y ampliar las funcionalidades de estas de manera progresiva. La puesta en práctica de este sistema no sólo beneficia a especialistas y demás usuarios del software, sino también a todos los clientes que utilizan los servicios que ofrecen las aplicaciones que son actualizadas. Para cumplir con lo anteriormente dicho, se han capturado los requisitos funcionales partiendo de la necesidad real de un mecanismo de actualización automática.

2.2.1 Requisitos funcionales del sistema

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir y que definen el comportamiento interno del software. Estos describen los servicios que se espera que el sistema cumpla para satisfacer las necesidades del usuario.

RF1. Verificar permisos de ejecución.

RF2. Modificar parámetros de acceso al repositorio.

RF3. Verificar permisos de lectura y escritura en los directorios a utilizar.

RF4. Obtener lista de aplicaciones soportadas.

RF5. Descargar el fichero Recurso de Versiones del repositorio.

FR5. Notificar nuevas actualizaciones disponibles.

FR5. Determinar lista de actualizaciones posibles a ejecutar.

RF6. Mostrar lista de actualizaciones posibles a ejecutar.

RF7. Seleccionar las actualizaciones a ejecutar.

RF8. Descargar el Paquete de Actualización.

RF9. Iniciar la actualización.

RF10. Mostrar progreso de las tareas.

RF11 Pausar operación.

RF12 Reanudar operación.

RF13 Cancelar la actualización.

RF14 Cancelar la operación de restablecer.

RF15 Deshacer la última acción (Rollback).

RF16 Mostrar lista de versiones a restablecer.

RF17 Restablecer o revertir la actualización (Restore).

RF18 Eliminar directorios y archivos temporales.

RF19 Generar reporte de actualización.

RF20 Mostrar reporte de actualización

RF21 Informar al usuario una vez terminada la actualización.

2.2.2 Requisitos no funcionales del sistema

Usabilidad

RNF1. El sistema está diseñado para ser utilizado por usuarios con pocos conocimientos informáticos, facilitando mediante interfaces amigables que los usuarios adquieran las habilidades necesarias para explotarlos en un periodo de tiempo breve.

Seguridad

RNF2. Se mantendrá seguridad y control a nivel de usuarios de sesión, garantizando el acceso a la ejecución del actualizador sólo desde una cuenta de administración de la PC según los niveles establecidos de acuerdo a la función que realizan. Las contraseñas podrán ser cambiadas sólo por el propio usuario o por el administrador del sistema.

RNF3. El sistema genera un reporte donde muestra datos de las actualizaciones efectuadas como: usuario, versión anterior, versión actual, fecha y hora en que se realizaron las últimas 40 actualizaciones.

RNF4. El sistema permitirá el restablecimiento de la configuración anterior en caso de errores durante la actualización a partir de los respaldos o salvas realizadas de cada uno de los ficheros modificados.

RNF5. Si el usuario autenticado en el sistema operativo no tiene permisos de administración el Actualizador Automático aparecerá inactivo, es decir se le negará el acceso al mismo.

RNF6. El sistema podrá ser utilizado en todo momento.

RNF7. Se verificará en cada paso del proceso de actualización los permisos de acceso, lectura y escritura sobre los ficheros y carpetas.

Eficiencia

RNF8. El sistema deberá ser rápido ante la solicitud de una actualización, mostrando un progreso rápido en cada una de las tareas iniciadas. Para ello se respetará las buenas prácticas de programación que incrementan el rendimiento en operaciones costosas y reducen el tiempo de respuesta para cada solicitud al mínimo posible.

RNF9. En las estaciones de trabajo es donde estará disponible la solución, las que necesitan capacidad de hardware que soporten la ejecución de hilos de procesos. Por lo que se escogieron estaciones de trabajo de 256 MB de memoria RAM y un microprocesador de 2.0 Hz con Sistema operativo Windows.

Soporte

RNF10. Se permitirá realizar copias de seguridad de los archivos hacia otro dispositivo de almacenamiento externo, además de recuperar los mismos partir de los respaldos realizados.

RNF11. Se permitirá el chequeo de las operaciones y acceso de los usuarios a la aplicación.

RNF12. Una vez terminado el software, se realizarán las pruebas al mismo, además de una capacitación del personal responsable de la aplicación y mantenimiento de software.

RNF13. Se hace necesario que las aplicaciones externas que deseen actualizarse con el Actualizador Automático creen en el registro de Windows una clave con el nombre InfoUpdater y dentro los valores AppUpdaterName, AppUpdaterPath y AppUpdaterVersion.

Restricciones de diseño

RNF14. Para el desarrollo se utilizará C# como leguaje de programación.

RNF15. Se utilizará Visual Studio como entorno de desarrollo.

RNF16. Para realizar el modelado del software se utilizó la herramienta Enterprise Architect predefinida por el Departamento de Sistemas de Apoyo a la Salud.

RNF17. La capa de Aplicación contendrá todas las vistas y la lógica de la presentación, además del controlador de instancia única para la aplicación.

RNF18. La capa Motor de descargas contendrá las reglas del negocio que deben cumplirse, en correspondencia con el proceso de descarga de recursos desde un repositorio.

RNF19. La capa Motor de Tareas contendrá las reglas del negocio que deben cumplirse, en correspondencia con el proceso de ejecución y control de tareas descritas en ficheros XML.

Requisitos para la documentación de usuarios en línea y ayuda del sistema

RNF20. Se dispondrá de la documentación del sistema realizada haciendo uso de CMMI como modelo y RUP como metodología de desarrollo.

Interfaz

RNF21. Las interfaces de usuario del sistema serán diseñadas haciendo uso del completo conjunto de controles para el diseño de formularios que brinda Visual Studio y la librería *Qios.DevSuite.Components.DLL*, con un diseño de componentes muy parecido a los utilizados por el

paquete en la creación del paquete office. Cada uno de estos controles tiene asociado propiedades, métodos y eventos que permite desarrollar fácil y rápidamente las soluciones deseadas.

Requisitos de Licencia

RNF22. Se desarrollará el sistema sobre el Framework 2.0 de .NET, para garantizar que el mismo corra sobre MONO 2.2 y sea libre de licencias.

Requisitos Legales, de Derecho de Autor y otros

RFN23. El sistema será liberado bajo la marca comercial “alas”.

Estándares aplicables

RFN24. Para garantizar la calidad del software se hará uso de CMMI como modelo, para alcanzar el nivel 2 de madurez, generando los artefactos definidos por el área REQM con RUP como metodología y el lenguaje UML 2.0 para visualizar, especificar, construir y documentar los artefactos del sistema.

RNF25. Se utilizará el Estándar de Codificación definido por el Grupo Implementación de Negocio del Departamento de Sistemas de Apoyo a la Salud en su Versión 1.0.

2.2.3 Definición de los actores del sistema

Actores del sistema	Funciones
Administrador	Personal encargado de iniciar y asistir el flujo de eventos que comprende una actualización automática.
Sistema Operativo	Representa al sistema operativo.

Tabla 2.1 Actores del sistema

Sistema Operativo: Representa al sistema operativo que es el encargado de iniciar el caso de uso “Verificar Permisos de Ejecución”.

Administrador: Es la persona encargada de iniciar y asistir el flujo de eventos que comprende una actualización automática. El mismo tiene como condición que debe ser un administrador del equipo para poder interactuar con el sistema. Esta autorizado a ejecutar todas las funcionalidades del Actualizador Automático.

2.2.4 Diagrama de casos de uso del sistema

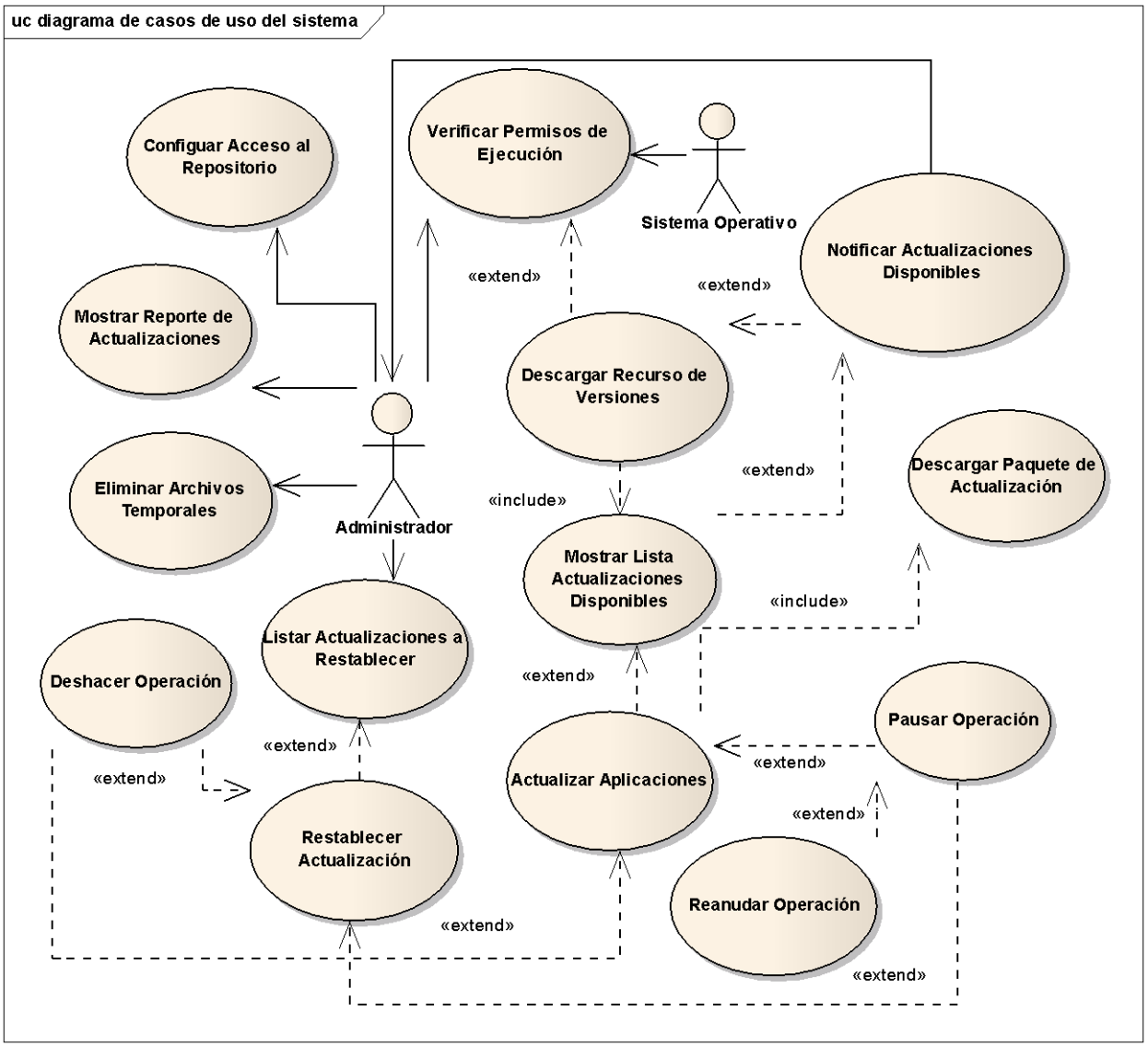

Figura 2.2 Diagrama de casos de uso del sistema

2.2.5 Especificación de los casos de uso arquitectónicamente significativos

- **Configurar Acceso al Repositorio**

Configurar Acceso al Repositorio	
Objetivo	Establecer una configuración para el acceso al repositorio y descarga especificando cada una de las variables necesarias.
Actores:	Administrador
Resumen:	El caso inicia cuando el actor accede a la opción “Configurar repositorio”. Este modifica las variables necesarias para hacer la conexión con el repositorio, el actor acepta o cancela la configuración. El caso de uso termina.
Precondición:	No existe.
Postcondiciones	No existe.

Tabla 2.2 Especificación del Caso de Uso. Configurar Acceso al Repositorio

- **Descargar Recurso de Versiones**

Descargar Recurso de Versiones	
Objetivo	Descargar el fichero XML Recurso de Versiones hacia una dirección local de la PC.
Actores:	
Resumen:	El caso de uso inicia cuando el sistema accede a la configuración del repositorio, luego verifica los permisos de lectura y escritura en la dirección destino del fichero para realizar la conexión y descargar hacia dicha dirección el fichero RecursoVersiones.Xml. El caso de uso termina.
Precondición:	El caso de uso Verificar Permisos de Ejecución fue ejecutado antes de este.
Postcondiciones	Descargar el fichero Recurso de Versiones completo.

Tabla 2.3 Especificación del Caso de Uso. Descargar Recurso de Versiones

- **Descargar Paquete de Actualización**

Descargar Paquete de Actualización	
Objetivo	Descargar el compactado Paquete de Actualización hacia una dirección local de la PC.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el sistema accede a la configuración del repositorio, luego verifica los permisos de lectura y escritura en la dirección destino del paquete para realizar la conexión y descargar hacia dicha dirección el compactado Paquete de Actualización. El caso de uso termina.
Precondición:	El caso de uso Verificar Permisos de Ejecución fue ejecutado antes de este.
Postcondiciones	Descargar el Paquete de Actualización completo.

Tabla 2.4 Especificación del Caso de Uso. Descargar Paquete de Actualización

CAPÍTULO 3. MOSTRAR LISTA DE ACTUALIZACIONES DISPONIBLES

Mostrar Lista de Actualizaciones Disponibles	
Objetivo	Mostrar un listado con la actualización disponible para cada una de las aplicaciones soportadas en correspondencia con la actual versión en uso.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor es notificado de la existencia de nuevas actualizaciones y selecciona la opción Aceptar o cuando el actor ejecuta la aplicación. El sistema realiza la descarga del fichero Recurso de Versiones para conformar una lista con las aplicaciones que tienen nuevas actualizaciones y mostrarlas al actor. El caso de uso termina.
Precondición:	La descarga del fichero Recurso de Versiones ha sido exitosa y existen aplicaciones instaladas soportadas por el sistema.
Postcondiciones	Mostrar una lista de las aplicaciones que tienen nuevas actualizaciones.

Tabla 2.5 Especificación del Caso de Uso. Mostrar Lista de Actualizaciones Disponibles

- **Actualizar Aplicaciones**

Actualizar Aplicaciones	
Objetivo	Actualizar las aplicaciones mostrando el progreso de las tareas comprendidas en el proceso.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor selecciona la opción “Actualizar” las aplicaciones seccionadas, el sistema muestra una barra de progreso con el avance de la misma además de una lista de los archivos que están siendo modificados. Una vez que la barra de progreso llegue al 100%. El caso de uso termina.
precondición:	Debe existir al menos una aplicación en las lista de actualizaciones disponibles. El actor debe haber seleccionado al menos una aplicación a actualizar para que el sistema muestre activa la opción de “Actualizar”.

Postcondiciones	Se actualizaron las aplicaciones seleccionadas.

Tabla 2.6 Especificación del Caso de Uso. Actualizar Aplicaciones

- **Restablecer actualización**

Restablecer actualización	
Objetivo	Restablecer la versión anterior a la que está en uso.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor selecciona la opción de “Restablecer”, el sistema comienza a deshacer cada una de las tareas que fueron ejecutadas durante la actualización. Una ve terminado el proceso se le informa al actor y este selecciona la opción de “Aceptar”. El caso de uso termina.
Precondición:	No se debe ejecutar el caso de uso Eliminar copias de seguridad sino no se podrá restablecer ninguna versión anterior. Haber seleccionado de una lista una o varias aplicaciones a las que se les pueda iniciar el proceso en caso que haya sido ejecutada la opción “Restablecer” fuera del hilo de procesos de una actualización.
Postcondiciones	Restablecer exitosamente la versión anterior de una aplicación luego de haber ejecutado su actualización.

Tabla 2.7 Especificación del Caso de Uso. Restablecer actualización

En el presente capítulo fueron explicados detalladamente los procesos asociados con los actualizadores automáticos. Al analizar estos procesos se evidencian las problemas que presentan las aplicaciones producto de su funcionamiento diario e imprudencias de lo usuarios. Es por ello que se diseña una aplicación que permita optimizar esos procesos. Se realizó el modelo del dominio identificando los conceptos fundamentales con que trabajan los usuarios y deber trabajar el sistema, y partir de ahí se definieron los requerimientos funcionales y no funcionales del sistema, los cuales obtuvieron su traducción en el Modelo de casos de uso. Como resultado de este análisis se deriva el diseño de una herramienta automatizada que mejore el funcionamiento de los procesos descritos anteriormente.

CAPÍTULO 4. ANÁLISIS Y DISEÑO DEL SISTEMA

En el presente capítulo se realiza una descripción detallada del sistema propuesto donde se describe la arquitectura propuesta y aprobada por el Departamento de Sistemas de Apoyo a la Salud, además se aborda el diseño de la solución, modelándose los artefactos necesarios que contribuyen a la implementación del sistema.

3.1 Descripción de la arquitectura

La arquitectura de software representa básicamente la estructura de un sistema y comprende los principales componentes del mismo, sus propiedades externamente visibles y las relaciones entre ellos. Constituye además un modelo comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes. El estilo arquitectónico en tres capas propone una estructura organizacional para los componentes de las aplicaciones.

La capa de presentación está compuesta por formularios Windows Form diseñados haciendo uso del completo conjunto de controles que brinda Visual Studio, además de la librería Qios.DevSuite.Components.DLL que presenta un grupo de controles con un diseño muy parecido a los utilizados en la creación del paquete Office. Cada uno de estos controles tiene asociado propiedades, métodos y eventos que validan y muestran los datos que el usuario provee y solicita en cada una de las operaciones que realiza permitiendo desarrollar fácil y rápidamente las soluciones deseadas.

La capa de negocio o capa de procesos de actualización está integrada por clases controladoras, que encierran la lógica del negocio del sistema. Es en ella donde residen los programas que se ejecutan, recibiendo las peticiones del usuario y enviando las respuestas tras el proceso. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de tareas, para solicitar el inicio de una nueva tarea y ser notificado al término de la misma.

La capa de tareas está constituida por componentes NAnt que son los encargados de manipular las solicitudes del inicio de una tarea y notificar a la capa superior sobre el estado de la misma (progreso o terminada). Su funcionamiento básico está descrito por el trabajo con ficheros XML haciendo uso de una

clase generada con la herramienta “Símbolo del sistema de Visual Studio 2005” a partir del esquema de Nant, que proporciona un acceso fácil a secuencias XML basadas en el esquema de NAnt.

Entre los patrones utilizados están los llamados patrones GRASP, que tuvieron una importante utilidad en el diseño realizado. A cada clase le fueron asignadas las tareas que podían realizar según la información que poseían, además de crear las instancias de otras clases en correspondencia con la responsabilidad dada, poniéndose de manifiesto los patrones Experto y Creador. Con esto se logró conservar el encapsulamiento ya que los objetos realizan lo que se les pide utilizando información que ellos poseen [19].

A continuación se presenta el diagrama de paquetes del sistema propuesto:

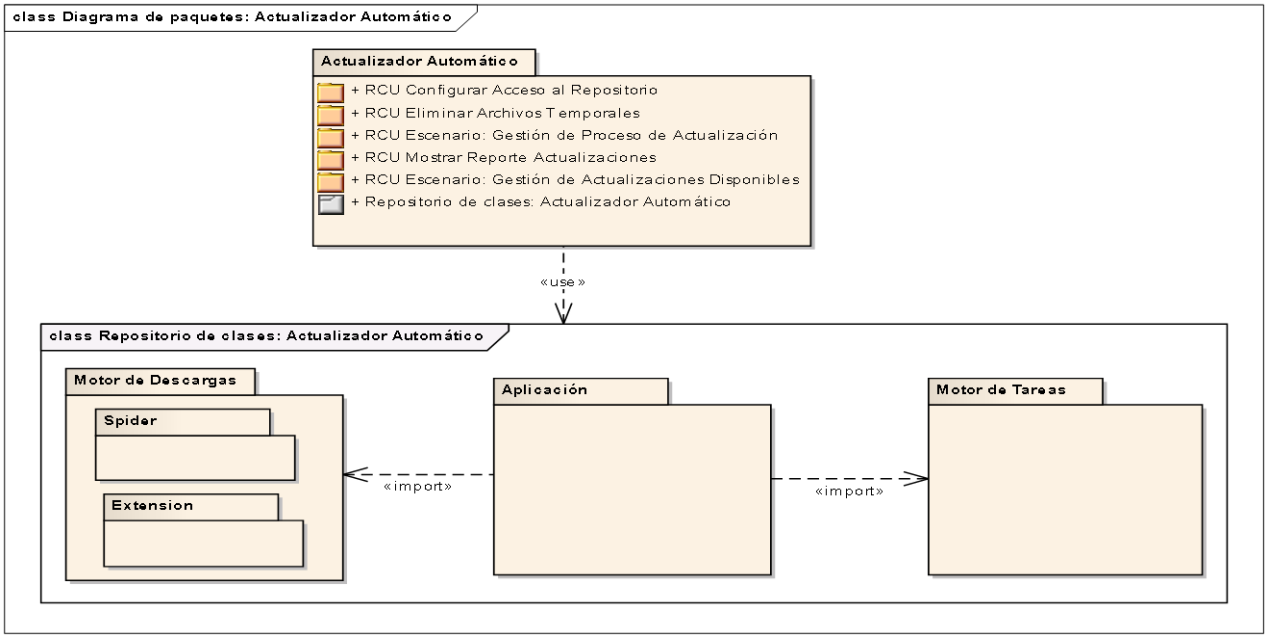

Figura 3.1 Diagrama de Paquetes

3.2 Modelo de análisis

En esta fase se refinan y estructuran los requisitos obtenidos con anterioridad, profundizando en el dominio de la aplicación lo que permitirá una mayor comprensión del problema para modelar la solución.

Los diagramas representados están organizados por escenarios para una mejor comprensión de los mismos.

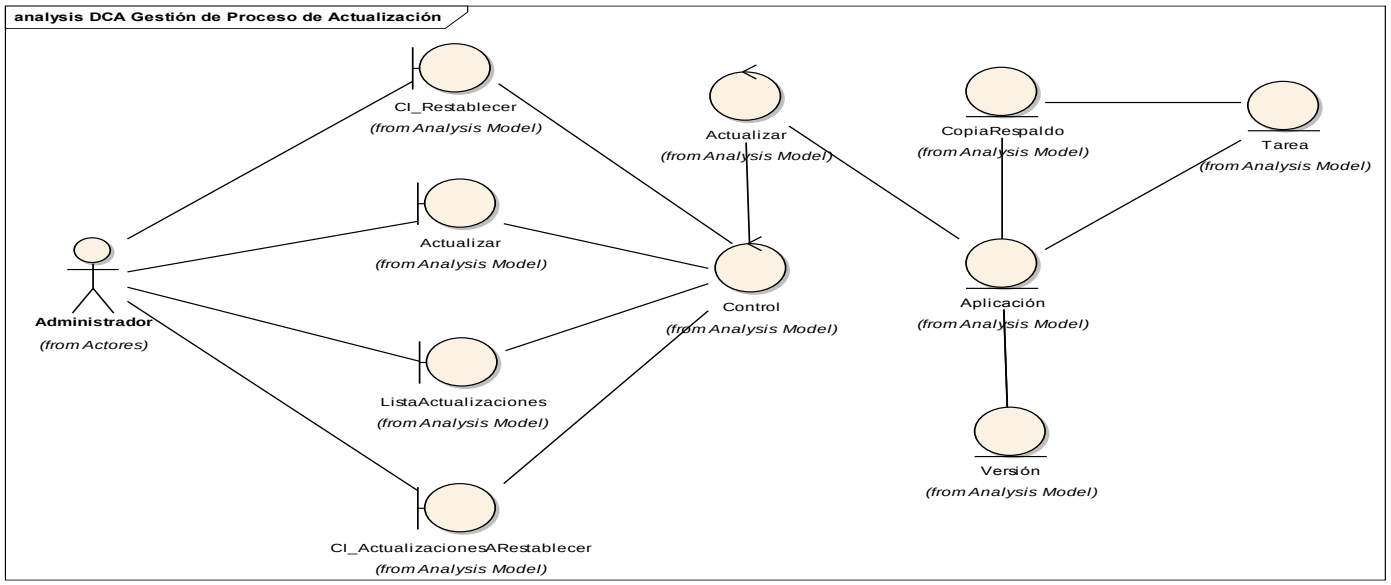

Figura 3.2 DCA Escenario: Gestión de Proceso de Actualización.

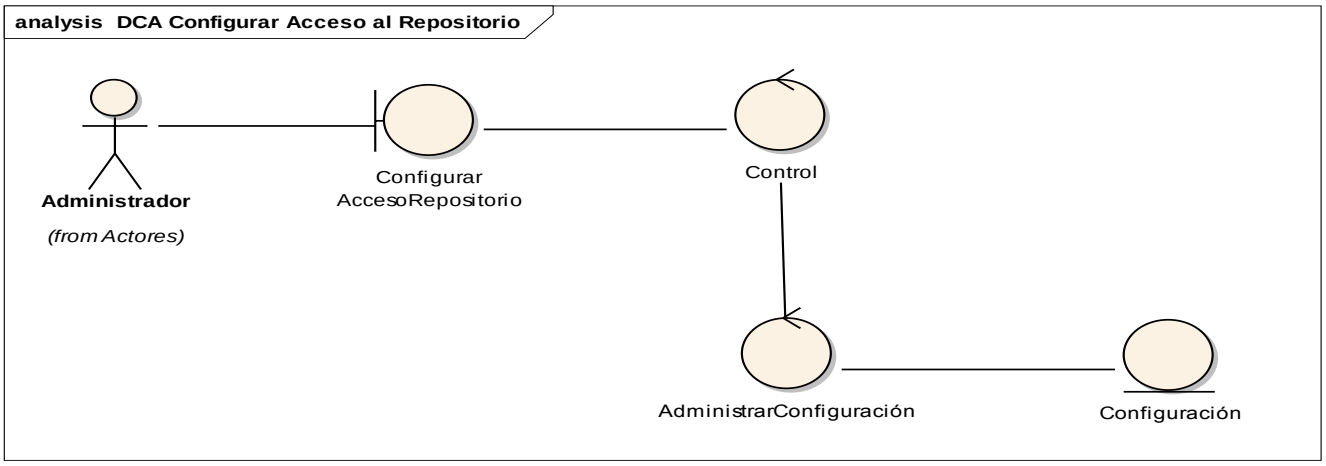

Figura 3.3 DCA: Configurar Acceso al Repositorio

3.3 Modelo de Diseño

El diseño es un refinamiento del análisis que tiene en cuenta los requisitos no funcionales, o sea, cómo cumple el sistema sus objetivos. Además debe ser suficiente para que el sistema pueda ser implementado

sin ambigüedades. Por otra parte tiene como propósito traducir los requisitos a una especificación que describe cómo implementar el sistema, es decir, pretende crear un plano del modelo de implementación.

Para la elaboración del diseño, generalmente se utilizan un grupo de patrones, o modelos para lograr los objetivos específicos. Estos forman parte de las mejores prácticas de investigadores y diseñadores de software orientado a objetos. De manera general, los patrones constituyen soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos, basados en la experiencia y que se ha demostrado que funcionan. [0].

Los patrones de diseño constituyen un vocabulario común de buenas soluciones, perfectamente identificadas y aplicables a distintos problemas típicos de diseño, que pueden encontrarse en diferentes contextos. Estos facilitan la reutilización del conocimiento experto como componentes de diseño y mejoran así la documentación, comprensión y comunicación del diseño final. Dentro de los más conocidos están los GRASP o patrones de asignación de responsabilidades.

Los patrones GRASP se utilizan con el objetivo de asignar responsabilidades a las diferentes clases que se definen en el diseño. Dentro de este grupo se identifican cinco patrones muy utilizados: experto, creador, alta cohesión, bajo acoplamiento y el controlador. [0]

En el diseño se modela el sistema para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Al mismo tiempo se define la arquitectura del sistema. En este modelo, los casos de uso son realizados por las clases del diseño y sus objetos, a partir de los cuales se forma el diagrama de clases del diseño.

Una *clase de diseño* es aquella suficientemente detallada que sirve como base para generar código fuente o lo que es lo mismo, es aquella cuya especificación es completa hasta un nivel que se pueda implementar. [0]

Los **diagramas de clases de diseño** exponen un conjunto de interfaces, colaboraciones y sus relaciones. Se utilizan para modelar la vista de diseño estática de un sistema. Estos son de gran importancia, ya que permiten visualizar, especificar y documentar modelos estructurales. Los diagramas de clases de diseño forman parte de las realizaciones de casos de usos.

Otro de los artefactos indispensables en la realización de los casos de uso son los **diagramas de interacción**, que muestran gráficamente como los objetos se comunican entre ellos a fin de cumplir con los requerimientos. Los diagramas de interacción juegan un papel importante en esta disciplina, ya que sirven para modelar los aspectos dinámicos de un sistema. [0]

Dentro de los diagramas de interacción, se definen dos tipos fundamentales, los de secuencia y colaboración, generalmente se recomienda utilizar las colaboraciones en el análisis y la secuencia en el diseño. Un **diagrama de secuencia** es un diagrama de interacción que destaca la ordenación temporal de los mensajes. [23]

Como se evidencia en el diagrama de paquetes, el sistema está dividido en 5 procesos: Gestión de Actualizaciones Disponibles, Gestión de Proceso de Actualización, Configurar Acceso al Repositorio, Eliminar Archivos Temporales y Mostrar Reporte Actualizaciones. Para cada proceso se modela un diagrama de clases del diseño y por cada escenario de dicho diagrama es modelado un diagrama de interacción. Como diagrama de interacción fue seleccionado el de secuencia.

Gestión de Proceso de Actualización

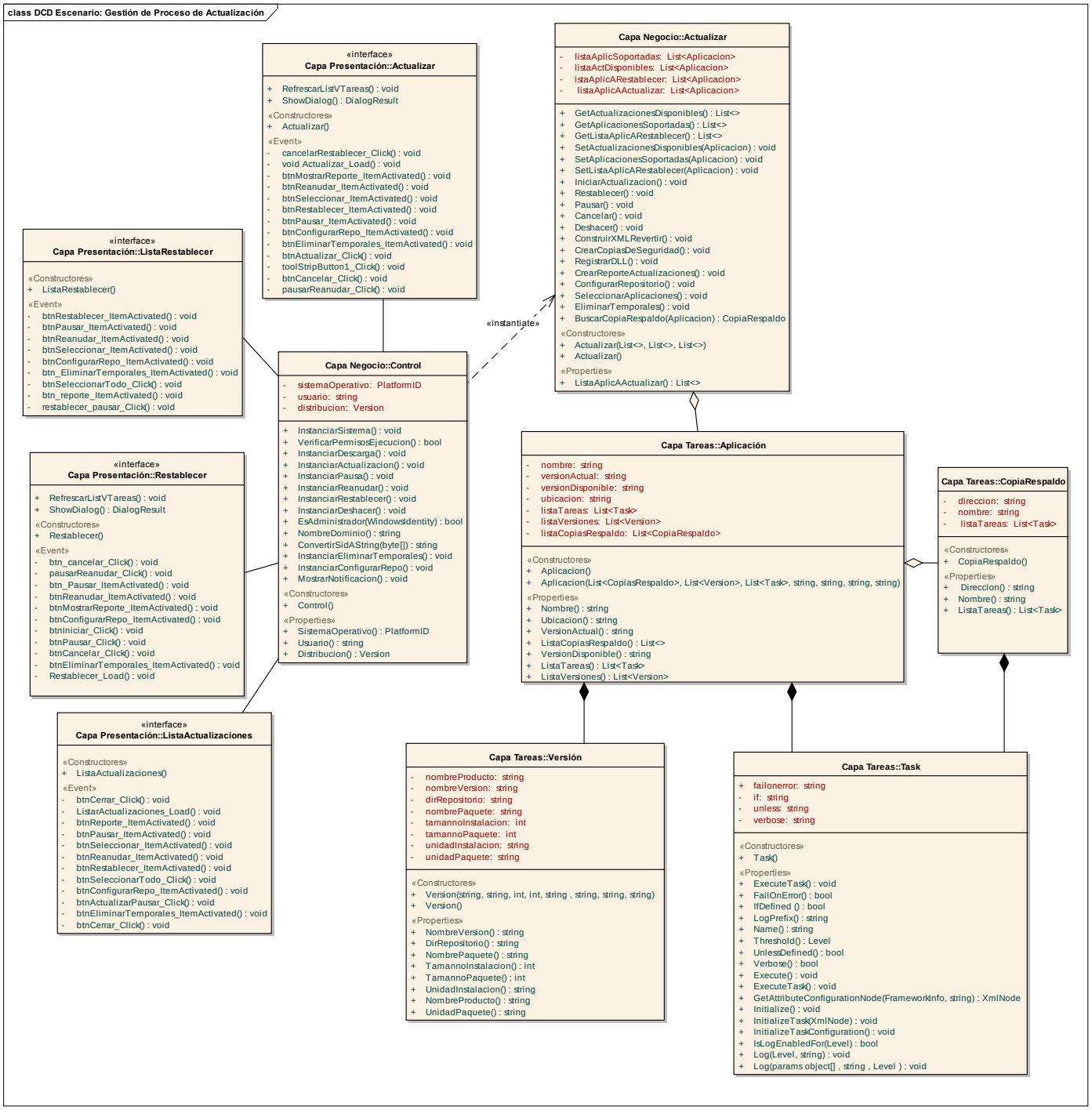

Figura 3.4 DCD Gestión de Proceso de Actualización

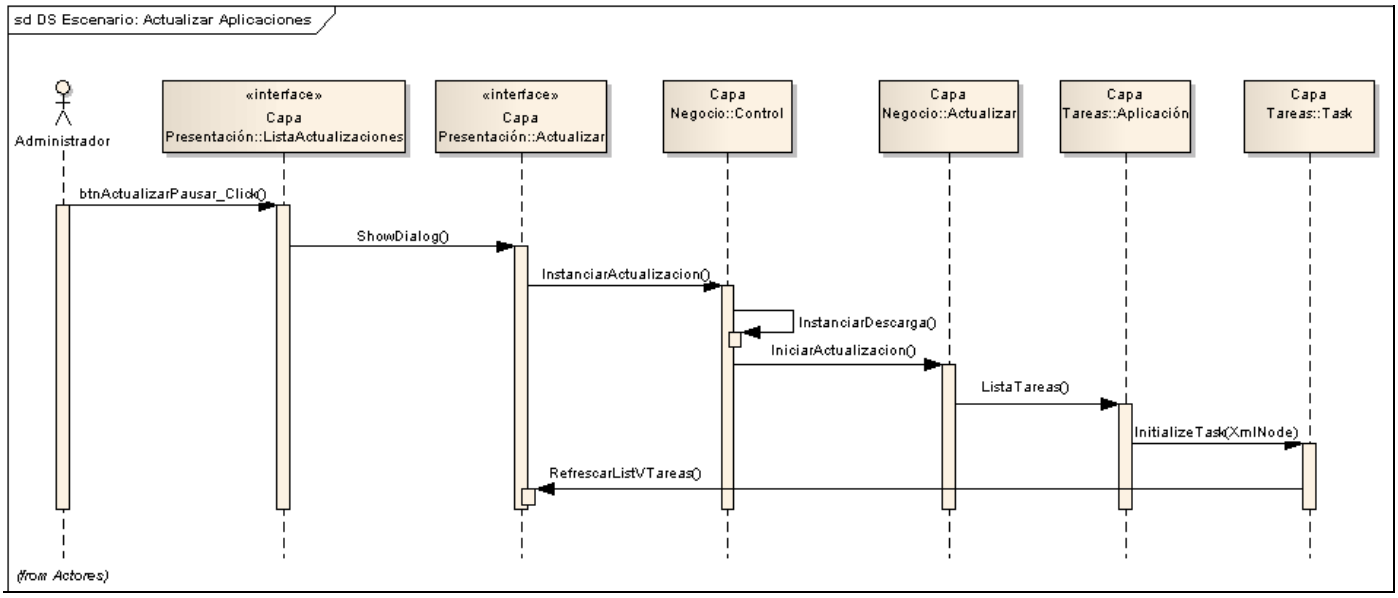

Figura 3.5 DS Escenario: Actualizar Aplicaciones

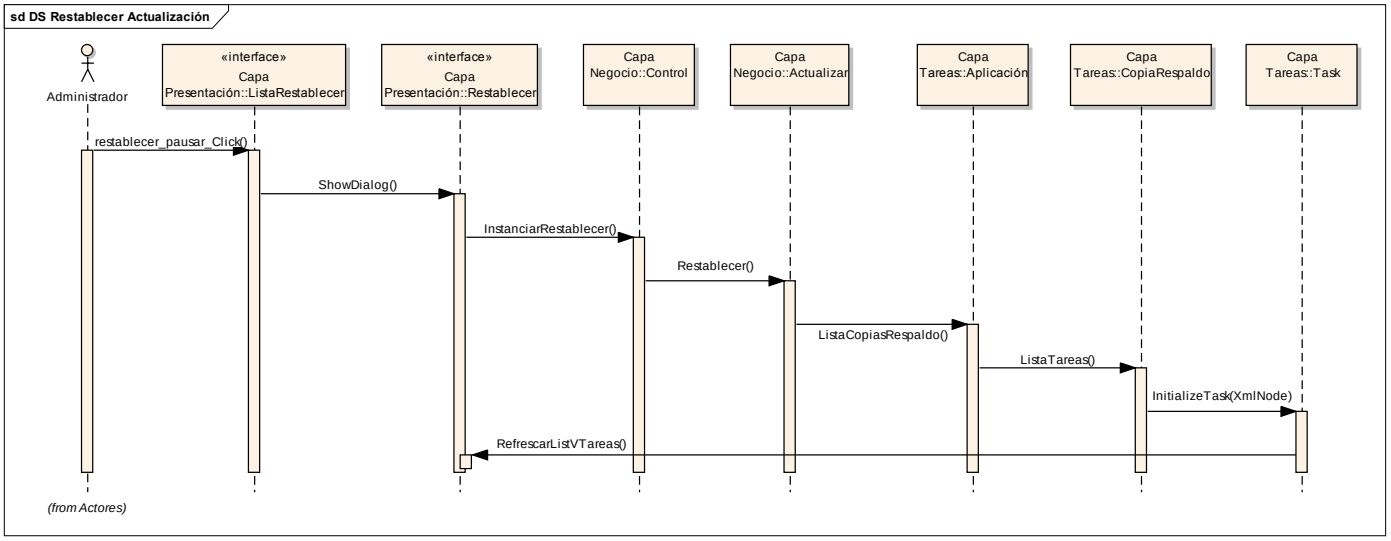

Figura 3.6 DS Escenario: Restablecer Actualización

Configurar Acceso al Repositorio

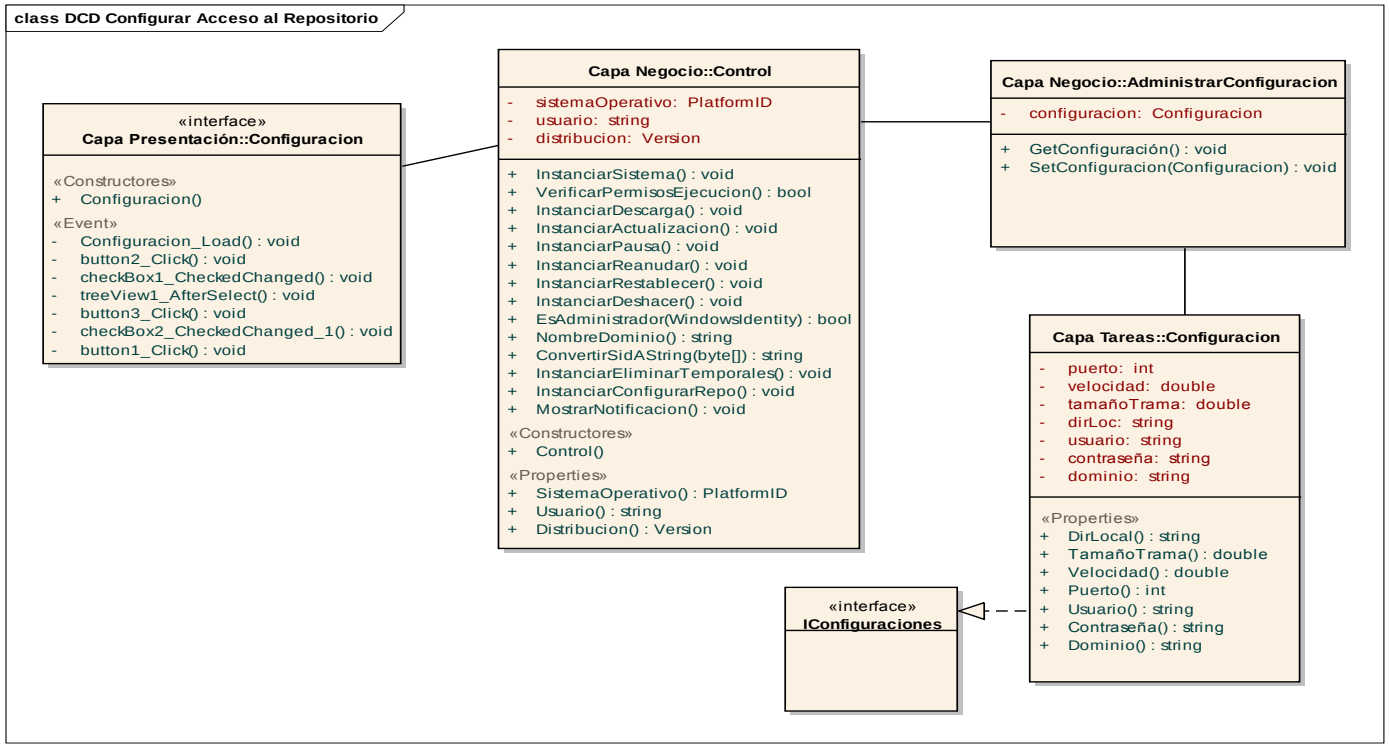

Figura 3.7 DCD Configurar Acceso al Repositorio

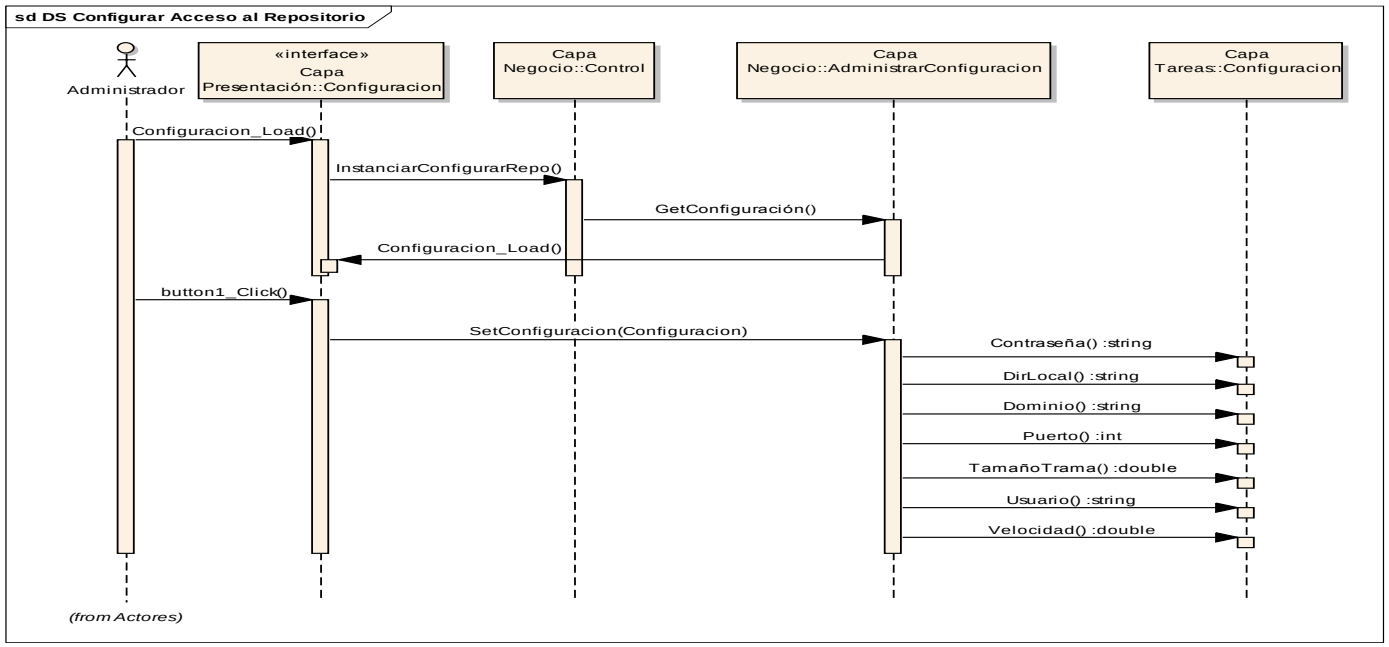

Figura 3.8 DS Configurar Acceso al Repositorio

Seguidamente serán explicadas algunas de las clases que han sido identificadas para su futura implementación describiéndose las responsabilidades que realizarán los formularios Windows Forms que responden a la Lógica de Negocio. De esta manera se tendrá una comprensión mayor del funcionamiento que tendrá el sistema en desarrollo.

3.3.1 Descripción de las clases

Nombre: formulario “Actualizar”
Descripción: Esta es la interfaz encargada de visualizar al usuario todas las tareas que están siendo ejecutadas durante el proceso de actualización. De estas tareas se muestran datos como: nombre del archivo o fichero que esta siendo modificado en la columna *Objeto*, tarea que esta siendo ejecutada la columna *Evento* y el estado del tarea (Iniciando, Ejecutando o Terminada) en la columna *Estado.*

Tabla 3.1 Descripción de la clase formulario Actualizar

Nombre: formulario “ListaActualizaciones”
Descripción: Esta es la interfaz encargada de visualizar la lista de las actualizaciones disponibles en el repositorio por cada una de las aplicaciones instaladas en la PC soportadas por el actualizador. Brinda además la posibilidad al usuario de seleccionar las actualizaciones que desea instalar y dar inicio al proceso.

Tabla 3.2 Descripción de la clase formulario ListaActualizaciones

Nombre: controladora “Actualizador”
Descripción: Es la clase encargada de ejecutar todas las acciones comprendidas dentro del proceso de actualización. Estas acciones pueden ser: Inicializar la ejecución de las tareas de actualización o restablecimiento, pausar las mismas, reanudarlas, realizar copias de seguridad, registrar componentes entre otras.

Tabla 3.3 Descripción de la clase controladora Actualizador

Nombre: controladora “Control”
Descripción: Es la clase encargada de instanciar y crean un hilo de procesos único para cada actualización garantizando que se ejecute cada una de las funcionalidades de la clase controladora Actualizar en su momento. Su función principal es controlar cada unos de los y los de procesos que levanta una actualización, además de verificar los requerimientos previos y secundarios de la misma.

Tabla 3.4 Descripción de la clase controladora Control

Como resultado del estudio realizado en este capítulo, correspondiente al flujo de diseño, se identificaron las fundamentales clases que deben ser definidas para que el sistema funcione satisfactoriamente, así como los atributos y métodos que deben tener las mismas para brindarle al desarrollador una idea clara de lo que se debe implementar. Además, se obtuvo la realización de casos de uso por procesos, donde se elaboraron los diagramas de clases y los diagramas de secuencia correspondientes.

CAPÍTULO 5. IMPLEMENTACIÓN

La fase de Implementación es la consecuencia del flujo de trabajo de Análisis y Diseño. En este capítulo se describen las clases y subsistemas implementados en términos de componentes. Se muestra el Diagrama de Despliegue como parte del Modelo de Implementación, que indica la distribución física de la solución implementada. Además, se proporciona una detallada explicación de dicha solución.

4.1 Modelo de implementación.

- ✓ El modelo de implementación describe cómo los elementos del diseño se implementan en componentes. Entre los componentes se puede encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe entre los paquetes y clases del modelo de diseño y los subsistemas y componentes físicos.
- ✓ El propósito del Modelo de Implementación es definir la organización del código, planificar las integraciones de sistema necesarias en cada iteración e implementar las clases y subsistemas encontrados durante el Diseño.
- ✓ Se debe proponer una estrategia de codificación que defina los formatos para la asignación de nombres a las variables, estilo de programación y métodos de documentación.[25]

4.1.1 Diagramas de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Estos muestran las opciones de realización como el código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes o bibliotecas cargadas dinámicamente. Estos tienen relaciones de traza con los elementos del Modelo de Diseño.

De los estereotipos estándar que se aplican a los componentes según el lenguaje de modelado UML se emplean: *library* y *file*, los cuales representan una biblioteca de objetos estática o dinámica y un documento que contiene código fuente, respectivamente.

Las relaciones de dependencia se utilizan en los diagramas de componentes para indicar que un componente se refiere a los servicios ofrecidos por otro componente. Los distintos componentes pueden agruparse en paquetes según un criterio lógico y con vistas a simplificar la implementación. Estos paquetes son estereotipados como <<subsistemas>>.

Cada subsistema puede contener componentes y otros subsistemas. La descomposición en subsistemas no es necesariamente una descomposición funcional. La relación entre paquetes y clases en el nivel lógico es el que existe entre subsistemas y componentes en el nivel físico.

A continuación se exponen los Diagramas de Componentes asociados a varios de los subsistemas de implementación identificados. Siguiendo la arquitectura en capas la estructuración en subsistemas de implementación es la siguiente:

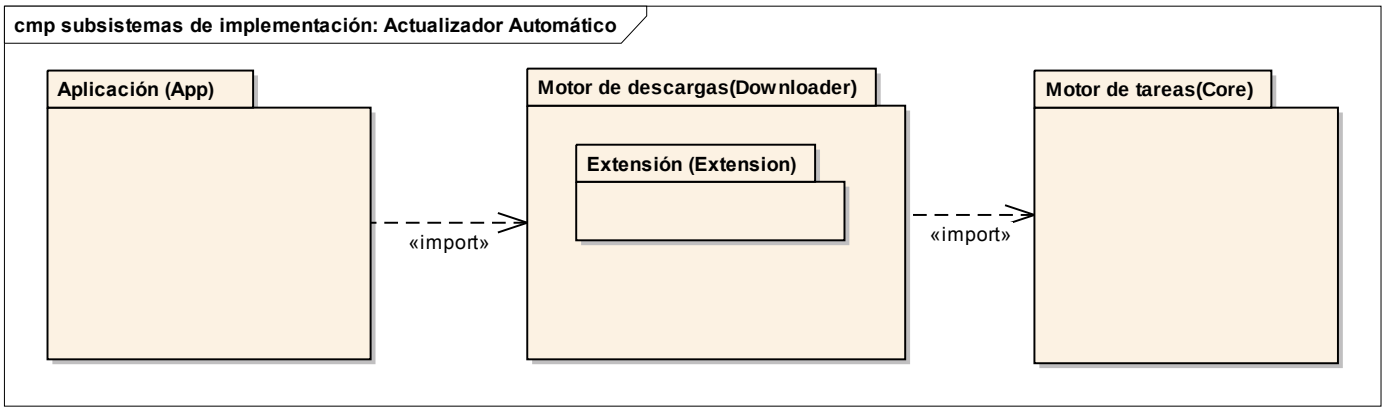

Figura 4.1 Subsistemas de implementación.

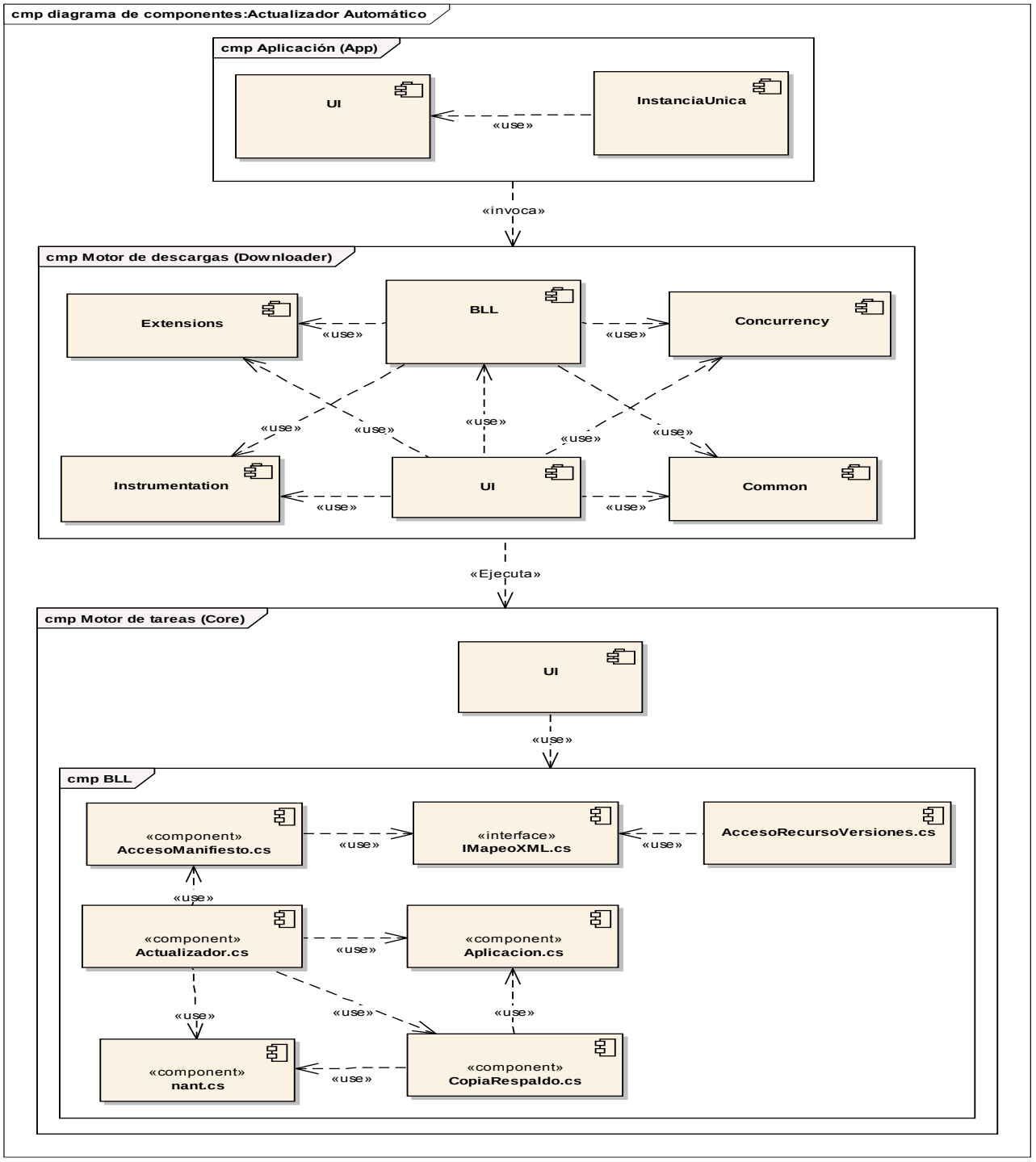

Figura 4.2 Subsistemas de implementación por capas.

4.1.2 Diagrama de despliegue

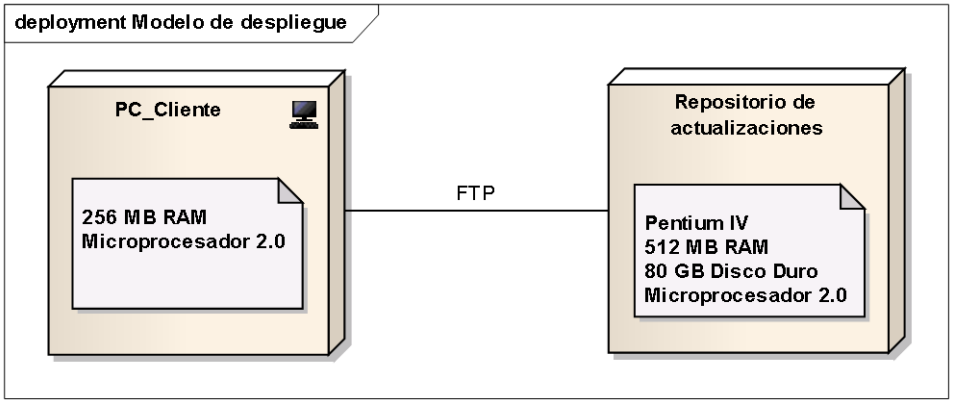

Figura 4.3 Diagrama de despliegue.

4.2 Tratamiento de errores

Durante el tiempo de ejecución de un sistema pueden fracasar diferentes rutinas; es esto a lo que comúnmente se le llama excepción. Mediante el tratamiento de excepciones se restaura un estado en el que la rutina pueda seguir la ejecución, lo que permite obtener un sistema más robusto y fiable.

En el sistema propuesto, el control de las excepciones se lleva a cabo a toda porción de código, donde pueda surgir alguna situación inesperada, especialmente donde se ejecutan sentencias que manipulan los datos que viajan desde y hacia los ficheros XML.

Para el manejo de las excepciones o errores, en las clases controladoras de procesos, se utiliza el bloque try para detectar cuando ocurra algún fallo y mediante el catch se manejarán dichas excepciones, mediante mensajes que se muestran en la interfaz de usuario. Entre las llaves de try se escribe el código que hará funcionar el programa. Para capturar la excepción que puede generar este código se necesita otra instrucción llamada **catch** (capturar).

4.3 Estrategias de codificación. Estándares y estilos a utilizar

La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede

modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento. Aunque la legibilidad y la mantenibilidad son el resultado de muchos factores, una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación sobre el que se efectuarán luego revisiones.

Si se aplica de forma continuada un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener**.**

Identación	
Objetivo	Lograr una estructura uniforme para los bloques de código así como para los diferentes niveles de anidamiento.
Inicio y fin de bloque	Se recomienda dejar dos espacios en blanco desde la instrucción anterior para el inicio y fin de bloque {}. Lo mismo sucede para el caso de las instrucciones if, else, for, while, do while, switch, foreach.
Aspectos Generales	El identado debe ser de dos espacios por bloque de código. No se debe usar el tabulador; ya que este puede variar según la PC o la configuración de dicha tecla. Los inicios ({) y cierre (}) de ámbito deber estar alineados debajo de la declaración a la que pertenecen y deben evitarse si hay sólo una instrucción. Nunca colocar ({) en la línea de un código

	cualquiera, esto requiere una línea propia.

Tabla 4.1 Estándar de codificación. Identación

Comentarios, separadores, líneas, espacios en blanco y márgenes		
Ubicación de comentarios	Al inicio de cada clase o función y al final de cada bloque de código.	Se recomienda comentar al inicio de la clase o función especificando el objetivo de la misma así como los parámetros que usa (especificar tipos de dato, y objetivo del parámetro) entre otras cosas.
Líneas en blanco	Se emplean antes y después de métodos, clases y estructuras.	Se recomienda dejar una línea en blanco antes y después de la declaración de una clase o de una estructura y de la implementación de una función.
Espacios en blanco	Entre operadores lógicos y aritméticos.	Se recomienda usar espacios en blanco entre estos operadores para lograr una mayor legibilidad en el código. Ejemplo: producto = nomproducto.
Aspectos generales	Sobre el comentario	Se debe evitar comentar cada línea de código. Cuando el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco. En caso de que se necesite comentar una sola instrucción se suprime la línea en blanco o se escribe a continuación de la

		instrucción
	Sobre los espacios en blanco	Después del corchete abierto y antes del cerrado de un arreglo. Después del paréntesis abierto y antes del cerrado. Antes de un punto y coma.

Tabla 4.2 Estándar de codificación. Comentarios, separadores, líneas, espacios en blanco y márgenes

Variables y constantes		
Apariencia de constantes	Todas sus letras en mayúscula	Se deben declarar las constantes con todas sus letras en mayúscula.
Aspectos generales	Nombres de las variables y constantes	El nombre empleado, debe permitir que con sólo leerlo se conozca el propósito de la misma.

Tabla 4.3 Estándar de codificación. Variables y constantes

Clases y Objeto		
Apariencia de clases y objetos	Primera letra en mayúscula.	Los nombres de las clases deben comenzar con una letra T en mayúscula, la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación PascalCasing*. Ejemplo: TMiClase (). Para el caso de las instancias se comenzara con un prefijo que identificara el tipo de

		dato, este se escribirá en minúscula.
Apariencia de atributos	Primera letra en minúscula	El nombre que se le da a los atributos de las clases debe comenzar con la primera letra en minúscula, la cual estará en correspondencia al tipo de dato al que se refiere, en caso de que sea un nombre compuesto se empleará notación CamellCasing**.
Apariencia de las funciones	Primera letra en mayúscula	Para nombrar las funciones se debe tratar de utilizar verbos que denoten la acción que hace la función. Se empleará notación PascalCasing*. Ejemplo: function BuscarUnidad (). Si son funciones que obtienen un dato se emplea el prefijo get y si fijan algún valor se emplea el prefijo set.
Aspectos generales	Sobre las clases, los objetos, los atributos y las funciones.	El nombre empleado para las clases, objetos, atributos y funciones debe permitir que con sólo leerlo se conozca el propósito de los mismos.

Tabla 4.4 Estándar de codificación. Clases y Objeto

Controles		
Apariencia de los controles	Los controles tendrán un prefijo	El nombre que se le da a los

	para el tipo de datos en minúscula.	controles deben comenzar con las primeras letras en minúscula, las cuales identificaran el tipo de datos al que se refiere (ver tabla 1.2), en caso de que sea un nombre compuesto se empleará notación CamellCasing**. Ejemplo: btnAceptar

Tabla 4.5 Estándar de codificación. Controles

Nombrado de variables según tipo		
Tipo Datos	Prefijo	Ejemplo
int	i	iCantPacientes
flota	f	fPesoPaciente
double	d	dPesoCarro
bool	b	bPacienteActivo
string	s	sNombrePaciente
char	c	cLetra
De tipo enum	ev	evSexo
byte	b	bCantDiasPaciente
sbyte	sb	sbEdadPaciente
short	sh	shVariableShort
ushort	us	usVariableUshort
uint	ui	uiVariableUint
long	l	lVariableLong
ulong	ul	ulVariableUlong
decimal	dc	dcVariableDecimal
Objetos	o	oPacienteHistorico
Objetos de tipo Struct	st	stUnaStruct

Tabla 4.6 Estándar de codificación. Nombrado de variables según tipo

Nomenclatura de los controles visuales		
Control	Prefijo	Ejemplo
Botón	btn	btnAceptar
Etiqueta	lbl	lblNombre
Lista/Menú	mn	mnPrincipal
Campo de Texto	txt	txtFecha
Botón de Opción	bpt	optSexo
Casilla de Verificación	chx	chxBorrar
Casilla de Selección	cbx	cbxSexo

Tabla 4.7 Estándar de codificación. Nomenclatura de los controles visuales

En el capítulo se detallaron los diagramas de despliegue y de componentes, que muestran la distribución física del sistema y las dependencias lógicas entre los componentes de la aplicación. Se estructuraron las clases del diseño en paquetes y subsistemas de implementación. Se describieron los principios de diseño seguidos, concepción del tratamiento de errores y principios de codificación. En general, fue implementado el sistema en términos de componentes con el objetivo de dar solución a los requisitos especificados.

CONCLUSIONES

Con el desarrollo del presente trabajo se arriba a las siguientes conclusiones:

- ✓ El estudio realizado a cerca de los principales sistemas que a nivel mundial gestionan los procesos asociados a las actualizaciones automáticas, evidenció que estos no satisfacen los requerimientos necesarios que demandan la actualización de las soluciones alas.
- ✓ Se identificaron las principales acciones en términos de procesos que son llevadas a cabo por un actualizador automático para completar el flujo de eventos comprendidos en una actualización.
- ✓ La aplicación desarrollada contiene las principales funcionalidades asociadas al dominio de los actualizadores automáticos, logrando así un sistema acorde a las tendencias actuales.
- ✓ La aplicación creada elimina la introducción de errores en el proceso de actualización, garantizando la ejecución del mismo a través de transiciones por estados seguros y reversibles.
- ✓ Se asimiló la arquitectura definida la cual contribuyó al desarrollo de un sistema robusto y flexible.
- ✓ La solución desarrollada constituye una referencia para el desarrollo de sistemas similares y establece la base para la extensión de los productos desarrollados en el Centro de Informática Médica.

De esta forma, se cumplió con los objetivos trazados para la elaboración del trabajo de diploma y se desarrolló una aplicación informática que permite mejorar los procesos de actualización de soluciones informáticas.

REFERENCIAS BIBLIOGRÁFICAS

[1] Genevieve Sovereign. (06 de 09 de 2006). The Code Project. Recuperado el 18 de 01 de 2010, de http://www.codeproject.com/KB/applications/updater.aspx

[2] Chuletas de Informática de Universidad. (09 de 05 de 2007). Xuletas. Recuperado el 20 de 01 de 2010, de http://www.xuletas.es/ficha/conceptos-basicos-de-computacion/

[3] DEFINICION.DE. (s.f.). DEFINICION.DE.COM. Recuperado el 25 de 01 de 2010, de http://definicion.de/automatico/

[4] Morena, M.-A. S. (24 de 11 de 2008). Connexions. Recuperado el 12 de 02 de 2010, de http://cnx.org/content/m17404/latest/

[5] DELGADO, ANTONIO. 2009. CONSUMER EROSKI. CONSUMER EROSKI. [En línea] Fundación EROSKI, 25 de agosto de 2009. [Citado el: 13 de febrero de 2010.] http://www.consumer.es/web/es/tecnologia/software/2009/08/24/187121.php

[6] Genotrance. (06 de 04 de 2007). AppSnap. Recuperado el 27 de 01 de 2010, de http://appsnap.genotrance.com/

[7] Creative Commons Attribution 3.0 United States License. 2010. Nabber.org. Nabber.org. [En línea] CC-GNU GPL, 18 de febrero de 2010. [Citado el: 10 de enero de 2010.] http://www.nabber.org/.

[8] Maldonado, Daniel M. (28 de Septiembre de 2007). El CoDiGo K. Recuperado el 11 de 02 de 2010, de Arquitectura de programación en 3 capas: http://www.elcodigok.com.ar/2007/09/arquitectura-de-programacion-en-3-capas/

[9]. Kiccillof Nicolás, Reynoso Carlos. 2004. [En línea] marzo de 2004. [Citado el: 1 de enero de 2010.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf

[10] Garlan David, Shaw Mary. “An introduction to software architecture”. CMU Software Engineering Institute

[11] Universidad de las Ciencias Informáticas de Cuba. (01 de 08 de 2005). EVA. Recuperado el 07 de 02 de 2010, de http://eva.uci.cu

[12] Guía de Usuario de Enterprise Architect. [Online] [Citado: Enero 27,2009] Disponible en http://www.sparxsystems.com.ar/EAUserGuide/ea.html

[13] CORPORATION, M. (2010). MSDN , *Introducción a Visual Studio*. Recuperado el 13 de 02 de 2010, de http://msdn.microsoft.com/es-es/library/fx6bk1f4(VS.80).aspx

[14] Akif, Mohammad, y otros. Java y XML. s.l : Anaya.

[15] Creative Commons. (30 de 12 de 2006). Metodologías de desarrollo de software. Recuperado el 01 de 12 de 2009, de http://www.um.es/docencia/barzana/IAGP/Iagp2.html#BM1

[16] Mora, F. (27 de 12 de 2008). Lenguaje Unificado de Modelado. Recuperado el 25 de 01 de 2010, de http://www.dccia.ua.es/dccia/inf/asignaturas/GPS/archivos/Uml.PDF

[17] La Revista Informatica.com. (15 de 02 de 2009). Recuperado el 10 de 02 de 2010, de http://larevistainformatica.com/C1.htm

[18] *El Mundo Informático*. (8 de Mayo de 2007). Recuperado el 23 de Marzo de 2010, de PatronesGgraps. (Patrones de software para la asignación General de Responsabilidad). Parte II.: http://jorgesaavedra.wordpress.com/category/patrones-grasp/

[19] García, Joaquín. Patrones de diseño. 27 de Mayo de 2009. Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

[20] Saavedra, Jorge. PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad). Parte 2. Mayo 2007. Disponible en: http://jorgesaavedra.wordpress.com/category/patrones-grasp/

[21] Clases de diseño. Mayo 2009. Disponible en: http://www.taringa.net/posts/info/1492028/Workflow-De-Dise%C3%B1o,-Clases-de-Dise%C3%B1o,-Interfaces,-Diagrama.html

[22] Fernández, Ana. Diagramas de Interacción. Marzo 2001. Disponible en: http://tvdi.det.uvigo.es/~avilas/UML/node41.html

[23] Software y Aplicaciones Web . (14 de Febrero de 2008). Recuperado el 24 de Marzo de 2008, de UML Diagramas: http://www.jtentor.com.ar/post/UML-Diagramas.aspx

[24] Modelo de Implementación: Diagramas de Componentes y Despliegue. (s.f.). Recuperado el 15 de Marzo de 2009, de http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf

BIBLIOGRAFÍA

1. Sánchez González, C. (25 de 11 de 2008). *Aplicaciones en capas*. Recuperado el 29 de 01 de 2010, de http://oness.sourceforge.net/proyecto/html/ch03s02.html
2. Pressman Roger. Ingeniería del Software: *Un enfoque práctico*. Madrid, McGraw Hill, 2001.
3. Rumbaugh James, Michael, Premerlani William, Frederick Eddy y William Lorensen. *Object-orientedmodeling and design*. Englewood Cliffs, Prentice Hall, 1991.
4. CORPORATION, M. (2010). MSDN , *Introducción a Visual Studio*. Recuperado el 13 de 02 de 2010, de http://msdn.microsoft.com/es-es/library/aa291755%28VS.71%29.aspx
5. (s.f.). Recuperado el 05 de 01 de 2010, de http://www.vectorsf.com/servicios/soporte-y-mantenimiento-de-aplicaciones.
6. Larman Craig. UML y Patrones. 2a edición, Madrid, Prentice Hall.
7. (s.f.). Recuperado el 10 de 02 de 2010, de *Aplicaciones en capas:* http://oness.sourceforge.net/proyecto/html/ch03s02.html
8. (s.f.). Recuperado el 15 de 02 de 2010, de *Clear.com*: http://www.clikear.com/manuales/csharp/c10.aspx
9. Linux Online Inc. (02 de 07 de 2007). *Linux Questions.org*. Recuperado el 05 de 02 de 2010, de http://www.linux.org/info/
10. Maldonado, Daniel M. (29 de diciembre de 2009). *El CoDiGo K*. Recuperado 19 enero de 2010, de Archivos de la Categoría 'Proyecto MONO: http://www.elcodigok.com.ar/category/proyecto-mono/
11. Arenas, M. I. (s.f.). *Curso XML*. Recuperado el 13 de 12 de 2009, de http://geneura.ugr.es/~maribel/xml/introduccion/index.shtml
12. Autentia. (s.f.). Recuperado el 29 de 01 de 2010, de *arquitectura y soporte a desarrollo*: http://www.adictosaltrabajo.com/tutoriales/pdfs/grasp.pdf
13. CORPORATION, M. (2010). *Conceptos de Actualizaciones automáticas*. Recuperado el 13 de 02 de 2010, de http://technet.microsoft.com/es-es/library/cc781859%28WS.10%29.aspx
14. FULL SOLUTIONS SA . (s.f.). Recuperado el 30 de 11 de 2009, de *Full Solutions*: http://www.fullsolutions.com/licenciamiento/productos/visual-studio.aspx
15. *GetIt*. (09 de 10 de 2009). Recuperado el 27 de 11 de 2009, de http://puchisoft.com/GetIt/

16. IEEE Corporation. (8 de 12 de 2008). *IEEE Computer Society*. Recuperado el 09 de 12 de 2009, de http://www.computer.org/portal/web/swebok
17. Internelia. (2010). Recuperado el 18 de 02 de 2010, de *canalvisualbasic.net*: http://www.canalvisualbasic.net/manual-net/c-sharp/#cSharp
18. INTERSHARE, S.L. (20 de 11 de 2006). *Softonic*. Recuperado el 31 de 01 de 2010, de http://appsnap.softonic.com/
19. JoomlArt. (8 de 02 de 2010). *pressperu.com*. Recuperado el 15 de 02 de 2010, de http://www.pressperu.com/index.php?option=com_content&view=article&id=1648:microsoft-update-protege-la-pc-de-las-amenazas-actuales-de-seguridad&catid=80:internacional&Itemid=423
20. KC Softwares. (02 de 02 de 2010). Recuperado el 10 de 02 de 2010, de KC Softwares: http://www.kcsoftwares.com/index.php?sumo
21. Larman, C. “UML y patrones” Tomo I Capítulos 18, Páginas 185-215.
22. Larman, C. “UML y patrones” Tomo I Capítulos 18, Páginas 85-129.
23. Microsoft Corporation. (s.f.). CSharp-Online.NET. Recuperado el 13 de 02 de 2010, de http://es.csharp-online.net/C_sharp_NET_/_Cap%C3%ADtulo_1
24. Microsoft Corporation. (15 de 01 de 2008). Soporte Microsoft. Recuperado el 24 de 01 de 2010, de http://support.microsoft.com/kb/930858/es
25. Microsoft Corporation. (2010). Recuperado el 14 de 02 de 2010, de MSDN Visual Studio: http://msdn.microsoft.com/es-es/library/aa288436%28VS.71%29.aspx
26. Morena, M.-A. S. (24 de 11 de 2008). Connexions.com. Recuperado el 11 de 12 de 2009, de http://cnx.org/content/m17401/latest/
27. SIGHO. [En línea] [Citado el: 17 de febrero de 2010.] http://www.sigho.salud.gob.mx/descargas/pdf/manuales/actualizador_sighov3.3.0.pdf.
28. NANT. (s.f.). Recuperado el 07 de 12 de 2009, de NAnt : http://nant.sourceforge.net/
29. Paarmann, D. (09 de 07 de 2). Synaptic.com. Recuperado el 07 de 02 de 2010, de http://www.nongnu.org/synaptic/
30. Praag, J. v. (2006). AppGet. Recuperado el 30 de 01 de 2010, de http://www.app-get.com/whatisappget/
31. Reynoso, C. y. (2004). Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.

32. Teruel, P. A. (16 de 05 de 2001). Arquitectura de capas en Sistemas de Información. Recuperado el 11 de 02 de 2010, de http://www.ldc.usb.ve/~teruel/ci3715/clases/arqCapas2.html
33. UpdateStar Corporation. (2008). Recuperado el 17 de 01 de 2010, de UpdateStar: http://client.updatestar.com/
34. GOMEZ GALLEGO JUAN PABLO, ING. JORGE GALVES. 2007. Scribd. [En línea] 16 de septiembre de 2007. [Citado el: 18 de febrero de 2010.] http://www.scribd.com/doc/297224/RUP.
35. W3C ,MIT, ERCIM, Keio. (09 de 01 de 2008). Recuperado el 17 de 01 de 2010, de Guía Breve de Tecnologías XML: http://www.w3c.es/divulgacion/guiasbreves/tecnologiasXML
36. Zavala. (2000). Recuperado el 15 de 01 de 2010, de Ingeniería de Software: http://www.angelfire.com/scifi/jzavalar/apuntes/IngSoftware.html#POO
37. IEEE (1998) IEEE Std. 1219-1998, Standard for Software Maintenance, IEEE Computer Society Press, Los Alamitos, CA, 1998
38. Pigoski, T. M. (1997) Practical Software Maintenance – Best Practices for Managing Your Software Investment. John Wiley & Sons, New York, NY
39. Lientz, B.P. and Swanson, E.B. (1978). Characteristics of Application Software Maintenance. Communications of the ACM, June, 1978, pp. 466-471.
40. ISO/IEC (1999), ISO/IEC 14764, Software Engineering-Software Maintenance, ISO and IEC, 1999.
41. NAnt Corporation. (08 de 12 de 2008). NANT. Recuperado el 10 de 20 de 2009, de http://nant.sourceforge.net/