Universidad de las Ciencias Informáticas

Facultad 7

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

Título: Migración y extensión del Actualizador Automático para su despliegue en servidores UNIX.

Autores: Dayanna Hernández Pérez

Eddy Yanier Duque García

Tutores: Lic. Yasel Couce Sardiñas

Ing. Beatriz Fernández Carmenate

La Habana, junio del 2011

"Año 53 de la Revolución"

DATOS DE CONTACTO

Síntesis del Tutor: MSc. Yasel Couce Sardiñas

Correo electrónico: yaselc@uci.cu

Graduado de Licenciado en Ciencias de la Computación en el año 2005 en la Universidad Central de Las Villas. Posee Categoría Docente de Profesor Asistente. Ha impartido asignaturas de Sistemas Operativos y Seguridad Informática en la Facultad 7 desde el curso 2005 – 2006. Actualmente se desempeña como responsable del Departamento de Sistemas de Apoyo a la Salud.

Síntesis del Co tutor: Ing. Beatriz Fernández Carmenate

Correo electrónico: bcarmenate@uci.cu

Graduado de Ingeniero en Ciencias Informáticas, en el año 2010, en la Universidad de las Ciencias Informáticas. Se encuentra trabajando como especialista de calidad en el proyecto productivo Colaboración Médica.

DEDICATORIA

Dayanna:

A mis padres Hilda y Williams, a mi hermanita Rosi, a mis abuelos David, Zoila y Margot y a mi novio Jorge Humberto que son el mayor tesoro que me ha dado la vida, a ellos que han estado a mi lado durante toda mi carrera brindándome su apoyo incondicional les dedico este trabajo.

Eddy:

Hay personas que estimulan nuestros esfuerzos por la admiración que nos merecen. Son esas personas que nos hacen estar seguros de que ningún problema es grande porque se puede dividir entre dos, que ningún obstáculo es infranqueable porque se tiene una buena escalera. Son esas personas que cargan en su espalda nuestras desdichas y padecen en silencio sin esperar protagonismo cuando celebremos nuestras dichas. Esas personas son inigualables frente a nuestros ojos y en verdad lo son, porque tienen el poder de hacernos querer ser mejores personas para ser como ellas. A esas personas ofrendamos la vida si fuese necesario y acreditamos todo nuestro mérito porque con ellas siempre estaremos en deuda. A ustedes; mi incansable madre del alma, ejemplo de consagración laborar y que ha limpiado para mí el camino espinado hasta aquí y a mi linda abuela Hilda y segunda medre, ejemplo de cubana incorruptible y de compañera; dedico este modesto tributo.

AGRADECIMIENTOS

Dayanna:

A mis padres por su apoyo y confianza en todo momento, por todos los sacrificios que han hecho para proporcionarme un mejor futuro, por su cariño infinito, por el amor y las enseñanzas que me han dado. No alcanzarían los días de mi vida para agradecerles por ser los excelentes padres que son.

A mi hermanita querida por todo el amor y el cariño que siempre me ha dado, por estar ahí para decirme "Tata te amo", por ser mi principal fuente de inspiración en los momentos difíciles, por ser la persona por la cual quiero ser mejor cada día para que se sienta orgullosa de mí.

A mis abuelos por ser mis primeros padres, por darme todo el amor que se puede dar y más, por ser parte inseparable de mí.

A mi familia, que por ser tan grande no los puedo mencionar a todos, por confiar siempre en mí y por estar siempre a mi lado a pesar de la distancia.

A mi novio por compartir conmigo toda mi carrera, por estar a mi lado en los momentos difíciles, por ser la persona que me dice que sí puedo y me hace sentir que puedo lograr todo lo que me proponga.

A mis suegros y mi cuñada por todo el apoyo y el cariño que me han dado durante todo este tiempo, por quererme como una hija más.

A mi compañero de tesis por el excelente trabajo que ha realizado, por su dedicación a nuestro trabajo y por haberme hecho, a su modo, un lugar en su inmenso corazón.

A mis tutores Yasel y Beatriz y mi oponente Yubismel, por haberme guiado con pasos firmes que posibilitaron la realización de un trabajo de diploma con la calidad requerida.

A mis amigas Hildi, Yisi, Shirly, Mary, Nurin y Jaque por estar a mi lado en los momentos buenos y en los no tan buenos, por todas las cosas lindas que hemos compartido durante la carrera y por hacerme sentir que siempre puedo contar con ellas.

A los profesores que han contribuido a mi formación como persona e ingeniera.

A los chicos de mi primer año del 7106 que llegaron para quedarse en mi corazón Carlos, Boris, Richar y Dany.

A todos los que de una forma u otra han aportado su granito de arena, para que fuera posible la realización de este trabajo.

Eddy:

Hay personas que confían cuando nadie más cree, que están ahí cuando todos ya se han ido y que extienden su mano desinteresada cuando en el suelo creemos que todo esta perdido. Con esas personas sentimos una deuda eterna; a esas personas hoy quiero agradecer. A mis hermanos de mi amada tierra, que han estado para mí en las buenas y malas, que recargan y estimulan mis ganas de ser mejor persona y que no por lejos han tenido menos que ver en este resultado. A Grabiel, Ángel, Ernesto Duque, Yunier, Ernesto Samon, Dainier, Ramón, Carmen hoy quiero agradecer.

A los hermanos que he hecho en la universidad, que me han demostrado con creces su confianza, su admiración y su respeto incondicional, y que por ser los últimos que he hecho no han tenido menos que ver en este resultado. A Reinier, Andrés, Leandro, Melquiades, Raidel, Luismel, Arrebato, Guillermo, Franklin, José Carlos, Yoandris, Mojena, Alberto, Oniel hoy quiero agradecer. A mi compañera de tesis que ha sabido manejar certeramente mi carácter y de forma eficiente y decisiva ha contribuido a este resultado, sin ella nada de esto fuera posible. A Dayanna hoy quiero agradecer.

A los profesores que han confiado en mí y apostado a mi capacidad contribuyendo de forma decisiva a mi formación académica, base técnica de este resultado. Al Ing. Alexis Turruella, Ing. Luis Mariano Reina, Ing. Ernesto Sarduy hoy quiero agradecer. A mis tutores MSc. Yasel Couce Sardiñas e Ing. Beatriz Fernández Carmenate que han guiado mis esfuerzos en aras de este resultado, hoy quiero agradecer. A mi querida Facultad 7, motivo de mi afán y universo de mi quehacer hoy quiero agradecer. A mi universidad, motivo de mi orgullo hoy quiero agradecer.

A mi familia de sangre que ha limpiado para mí el camino espinoso de la vida. A mi linda Abuela Hilda, a mi abuela y abuelo Ángela y Gaspar, a mi luchadora madre Maramis, a mis hermanos y amigos Ernesto y Alexey, a mis tías Amancia, Marbellis y Maritza, a mis primas Danailis, Dianelis y Yuglia hoy quiero agradecer. A mi país que me dado la oportunidad de llegar hasta acá hoy quiero agradecer. Al comandante Fidel, a Mella, a Calixto García, a Maceo, a Céspedes, a Varela hoy quiero agradecer. Al apóstol que ha sido mi guía todos estos años y que me hace estar orgulloso de esta tierra y gritar en cualquier parte que soy cubano. A ustedes y a todos los que han tenido que ver con este resultado y que esta traicionera memoria no me deja recordar; a ustedes mi más sincera gratitud.

RESUMEN

El presente trabajo se realiza con el objetivo de desarrollar un sistema capaz de llevar a cabo las actualizaciones automáticas de las soluciones informáticas desarrolladas en el Centro de Informática Médica (CESIM); tanto en plataformas Windows, como UNIX.

Durante el desarrollo de este sistema se utilizó como metodología de software Proceso Unificado de Desarrollo (RUP, por sus siglas en inglés), el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) y Enterprise Architect 7.1 para el modelado del sistema. Con el objetivo de separar en tres capas las funciones básicas de la aplicación, se empleó el patrón de diseño Modelo-Vista-Controlador. Además se usó 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 es capaz de gestionar y ejecutar varias actualizaciones simultáneas, sin que esto interfiera en la calidad de la actualización; permitiendo al usuario seleccionar de forma interactiva acciones a ejecutar por el sistema sobre las actualizaciones en curso. Además cuenta con otras funcionalidades que posibilitan un servicio flexible de soporte y mantenimiento, al tiempo que establece costos de mantenimiento reducidos.El sistema obtenido en esta investigación constituye un impacto positivo sobre la estrategia de comercialización y despliegue de productos del CESIM.

Palabras Claves

Actualizaciones automáticas

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

INTRODUCCIÓN

La Informática, por su rapidez de crecimiento y expansión, ha transformado rápidamente las sociedades actuales; ocupando un papel fundamental en la mayoría de las esferas. Entre sus factores de surgimiento se destacaron, la necesidad de desarrollar nuevos métodos y medios eficaces para buscar, conservar y divulgar la información. Esto vino dado por la diversificación de las ramas científicas y creación de nuevas áreas de investigación, que hicieron más complejo su proceso de organización y suministro.

El siglo XX vino acompañado de grandes descubrimientos científicos y transformaciones tecnológicas, que ampliaron notablemente el conocimiento del hombre acerca del mundo que le rodeaba y a su vez, condujeron a cambios en la forma de interactuar con él. Así fue como poco a poco ocurrieron cambios en los soportes de la información, surgiendo las primeras computadoras.

Con el objetivo de administrar los recursos de hardware y software de las computadoras surgen diversos sistemas operativos o software de sistema. Hoy en día se pueden encontrar una gran variedad de estos; los hay de tipo comercial, como son los sistemas de la empresa Microsoft (Windows/2000/XP, MS-DOS), MacOS para Apple y los sistemas de libre distribución como es GNU/Linux.

En la actualidad existe una tendencia mundial a la migración al software libre y como principal exponente de éste, el sistema operativo Linux. Esto se debe a las numerosas ventajas que ofrece con respecto al software propietario, entre las que se pueden destacar: la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. (1)

Por otra parte, con el desarrollo creciente de la informática su campo de acción se ha ampliado cada vez más, a partir del hecho de que su objeto de estudio se halla presente en cada elemento de la vida. Es por esto que su aporte es fundamental en todas las esferas de la sociedad. La salud es uno de los campos más evolucionados y beneficiados con estos avances; en dicho sector uno de los aspectos más importantes y necesarios para brindar un adecuado servicio y atención médica con calidad, es la disponibilidad de la información.

El procesamiento de los sistemas de la salud con la utilización de las nuevas tecnologías de la información y las comunicaciones, permite la optimización de la mayoría de los procesos. De este modo se logra proporcionar un mejor servicio de salud y contribuir a mejorar la calidad de vida de la población.

Cuba no se encuentra exenta de estos avances tecnológicos y de las nuevas tendencias, cuenta con la Universidad de las Ciencias Informáticas (UCI); un centro estudiantil que logra vincular la docencia y la producción. Sus estudiantes están insertados en proyectos productivos, en los que se desarrollan software para las distintas esferas de la sociedad y a su vez, se encuentra inmersa en un proceso de migración hacia el software libre.

La UCI está compuesta por varias facultades, entre las que se encuentra la Facultad 7 en la que está ubicado el Centro de Informática Médica (CESIM), donde se crean aplicaciones informáticas destinadas al sector de la salud. Éstas 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.

Ante la aparición de los sistemas informáticos, surgen nuevas interrogantes que hoy preocupan a la comunidad desarrolladora. Uno de los grandes problemas que se afrontan actualmente en la esfera del desarrollo de soluciones informáticas, son las actualizaciones automáticas de las mismas, debido a que el mantenimiento del software es mucho más complejo que el mantenimiento del hardware.

Para dar solución a lo antes planteado, en el Departamento de Sistemas de Apoyo a la Salud perteneciente al CESIM, se desarrolló un sistema capaz de realizar las actualizaciones automáticas de los productos de dicho centro, el mismo fue desarrollado sobre el Framework Microsoft.NET. Este sistema presenta algunas dificultades como:

- ✓ No cuenta con un motor de descarga que tenga la implementación del protocolo CIFS, para que sea compatible con servidores de este tipo.
- ✓ No cuenta con un sistema de notificación para todos los procesos que se realizan sin la participación del usuario.
- ✓ No existe una fuerte integración con el Sistema Operativo, por lo que la aplicación no se inicia una vez que una sesión administrativa lo hace.

- ✓ No cuenta con un sistema de detección y eliminación de virus, por lo que no se garantiza que estos no se puedan infiltrar en el proceso de transferencia de archivos.

Partiendo de esto y considerando que un gran número de aplicaciones desarrolladas en el CESIM, están destinadas a ser desplegadas en servidores UNIX, los cuales en su mayoría no cuentan con interfaz gráfica; constituye un problema llevar a cabo las actualizaciones automáticas de estas soluciones en dichos servidores. Esto se debe a que el sistema existente está implementado sobre Framework Microsoft.NET el cual no es compatible con plataformas UNIX.

Surge así la necesidad de desarrollar un sistema capaz de gestionar y ejecutar las actualizaciones de software, tanto en plataformas Windows, como UNIX; garantizando de esta forma la operatividad de los software del sistema y manteniendo su funcionamiento libre de problemas. Éste debe establecer costos de mantenimiento reducidos al tiempo que proporcione actualizaciones del producto.

Actualmente en el CESIM no se cuenta con una herramienta que cubra estas necesidades y asegure la inexistencia de los problemas antes enunciados. La ausencia de un mecanismo que facilite la mejora continua de las soluciones alas sobre plataformas UNIX, constituye una gran limitante para brindar un servicio flexible de soporte y mantenimiento, que garantice el correcto funcionamiento de las aplicaciones. Por lo que no se cubren las necesidades de evolución y no se asegura la calidad del servicio prestado.

Dada la situación anterior, se define el siguiente **problema a resolver:** Incompatibilidad con plataformas UNIX del Actualizador Automático desarrollado en el Departamento de Sistemas de Apoyo a la Salud.

En correspondencia con el problema, el **objeto de estudio** está enfocado en el proceso de gestión de la actualización de las aplicaciones informáticas, de manera que el **campo de acción** se enmarca en el proceso de actualización automática de aplicaciones informáticas en servidores UNIX.

Para solucionar los problemas mencionados, se define como **objetivo general** del presente trabajo: Extender las funcionalidades del Actualizador Automático y migrarlo a un marco de trabajo que garantice su ejecución sobre plataformas UNIX.

Con la intención de alcanzar el objetivo propuesto se han trazado las siguientes **tareas a desarrollar**:

1. Analizar la primera versión del Actualizador Automático, detectando las deficiencias que presenta.
2. Realizar un análisis crítico del estado del arte de los intérpretes de comandos, evaluando las tendencias actuales en el mundo.
3. Obtener mediante el Proceso Unificado de Desarrollo los artefactos de los flujos de trabajo "Modelado de Negocio", "Requerimientos", "Análisis y Diseño" e "Implementación".
4. Diseñar la propuesta de solución, de manera que se garantice el uso de buenas prácticas en la implementación de la aplicación.
5. Definir la gramática para el compilador del intérprete de comandos.
6. Implementar el intérprete de comandos, para que se dote al actualizador con funcionalidades que garanticen su manipulación en servidores UNIX.
7. Migrar el Actualizador Automático a Mono, para que se garantice el despliegue de la herramienta en plataformas UNIX.

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 sistemas ya existentes vinculados al campo de acción y de la metodología, lenguajes y herramientas de desarrollo a utilizar.

Capítulo 2: Características del sistema: Se realiza un análisis del dominio de la aplicación y se definen los requisitos funcionales y no funcionales de la solución propuesta a la problemática planteada.

Capítulo 3: Análisis y Diseño del sistema: Modelación de la estructura de la aplicación a través de diagramas de clases del análisis y diseño. Además se muestra la interacción entre los actores y el sistema mediante los diagramas de secuencia.

Capítulo 4: Implementación: Se describe la implementación de las clases y subsistemas en términos de componentes de la solución propuesta, a través del Diagrama de componentes y el Diagrama de despliegue del sistema.

Capítulo 1. Fundamentación Teórica.

En el siguiente capítulo se abordarán un grupo de conceptos de vital importancia para el desarrollo de la investigación, los mismos serán referenciados en el resto del documento. Además se realizará una descripción de las distintas tecnologías que serán utilizadas para la creación del sistema; así como, los lenguajes y la metodología de software que servirá de guía para su formación estructural.

1.1 Conceptos asociados al dominio del problema.

Durante el desarrollo del trabajo de diploma se hace referencia a una serie de conceptos, que por no ser de uso común, es necesaria su previa comprensión para un mejor entendimiento de lo expuesto en la investigación.

En el ámbito de trabajo en el que se desarrolla el sistema una **Aplicación informática**: Es un tipo de software que permite al usuario realizar uno o más tipos de trabajo. Los procesadores de texto y las hojas de cálculo son ejemplos de aplicaciones informáticas. Las aplicaciones pueden haber sido desarrolladas a medida (para satisfacer las necesidades específicas de un usuario) o formar parte de un paquete integrado. (2) Estas aplicaciones informáticas son objeto de actualizaciones.

Se designa con el término **actualización informática** a aquella tarea o actividad que supone cambiar o alterar una aplicación por una versión más actual de la misma. (3)

Automático es aquello perteneciente o relativo al autómata. Este término proviene del griego ***automatos*** que significa “con movimiento propio” o “espontáneo”. (4) Por lo tanto, la noción de automático puede hacer referencia a distintas cuestiones; 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** 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 las

aplicaciones, pero también pueden contener actualizaciones de seguridad. Por todo esto se recomienda utilizar el actualizador automático de forma periódica, para asegurarse de que el sistema esté al día y tan seguro como sea posible. (5)

Un actualizador automático puede contar con dos módulos, el primero una herramienta desarrollada para el trabajo con interfaz gráfica y como segundo un **intérprete de órdenes o de comandos**. Un intérprete de comandos, es un programa informático que tiene la capacidad de traducir las órdenes que introducen los usuarios, mediante un conjunto de instrucciones facilitadas por él mismo. Las órdenes se introducen siguiendo la sintaxis incorporada por dicho intérprete. (6)

Una **línea de comandos** no es más que una cadena de caracteres que representan a un comando. Este comando corresponde a un archivo ejecutable del sistema o del shell, junto con otros argumentos opcionales (parámetros). (7)

1.2 Antecedentes.

Como primera aproximación, se puede decir que las actividades de mantenimiento del software son actividades de Ingeniería del Software, orientadas a la modificación o cambio del mismo por diferentes motivos. El cambio tiene como característica fundamental, el hecho de que primero se necesita una comprensión del objeto que se ha de cambiar, para poder hacer efectiva la modificación. Es conveniente que el software tenga una estructura y una documentación asociada que facilite su comprensión.

El software no se deteriora con el uso ni con el paso del tiempo, a diferencia de los materiales mecánicos que son producto de otras actividades de ingeniería, no sufre de un deterioro físico. No obstante, se suele considerar, que el software tiene un deterioro en su estructura cuando a lo largo del tiempo se van incluyendo numerosos cambios, que hacen que su estructura interna sea cada vez más difícil de entender. En ocasiones se ha denominado a este fenómeno como “erosión del diseño”. (8)

Las personas involucradas en la fase de mantenimiento de software esperan trabajar en estos defectos conocidos, ubicarlos y preparar un nuevo lanzamiento del software, conocido como un lanzamiento de mantenimiento, con el cual se espera resolver las fallas detectadas. Las operaciones de mantenimiento,

tienen lugar frente a la constante amenaza que implica la ocurrencia de una falla o error en un sistema y la necesidad de optimizar su rendimiento.

Los fabricantes de software, conocedores de que los atacantes están al acecho, corrigen los agujeros de seguridad y lanzan las respectivas actualizaciones. Por ello, es de vital importancia actualizar los sistemas, tanto el sistema operativo, como el resto de las aplicaciones de manera periódica. Aunque es posible actualizar los programas manualmente, lo más sencillo y seguro es hacerlo de manera automática; de forma que el propio sistema busque las actualizaciones, las descargue e instale sin la intervención del usuario.

Actualmente, la mayoría de los sistemas operativos disponen de un actualizador automático, que resulta muy útil para mantener sus servicios 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 empresa Microsoft y GNU/Linux, tuvieron gran aceptación y a su vez, despertaron el interés de otras compañías. Desarrollándose así, las más diversas soluciones para automatizar el proceso de actualización de software.

1.3 Estructura y funcionamiento de los intérpretes de comandos en sistemas UNIX.

Los sistemas encargados de llevar a cabo las actualizaciones automáticas han venido a facilitar este proceso para los usuarios, debido a que puede resultar un poco tedioso e inclusive considerarse una perdida de tiempo. Por esta vía se evita la necesidad de comprobar manual y periódicamente si la versión utilizada es la última disponible y por tanto la más segura, librando al usuario de intervenir en esta serie de pasos.

Con el objetivo de extender las funcionalidades del Actualizador Automático, para posibilitar su uso en servidores UNIX que no cuentan con interfaz gráfica, se hará un análisis de la estructura y funcionamiento de los intérpretes de comandos. Un intérprete de comandos es una interfaz entre el usuario y el sistema operativo, se hará referencia al mismo como "shell", para el caso de los sistemas UNIX. (9)

Para comprender su funcionamiento en los sistemas UNIX, es necesario entender su estructura. La cual se puede apreciar a través de la Fig.1: (9)

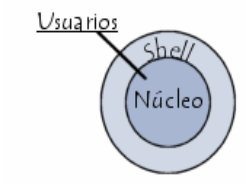

Fig.1 Estructura de los intérpretes de comandos en sistemas UNIX.

El núcleo es la parte del sistema operativo que sirve para interactuar con el hardware. Proporciona una serie de servicios que pueden ser utilizados por los programas, sin que estos tengan que preocuparse de cómo se gestiona el hardware. En general, el núcleo es el encargado de gestionar la memoria, mantener el sistema de archivos, manejo de las interrupciones, manejo de errores, realización de los servicios de entrada/salida y gestión de periféricos de entrada/salida. (10)

El shell por su parte actúa como un intermediario entre el sistema operativo y el usuario utilizando líneas de comando introducidas por dicho usuario. Su función consiste en la lectura de la línea de comandos, la interpretación de su significado, la ejecución del comando y luego la devolución del resultado a través de las salidas. El mecanismo de comunicación para asignar la salida estándar de un comando a la entrada estándar de otro para todos los sistemas UNIX son las tuberías (en inglés, "pipes"). (11)

Al ingresar la orden, el intérprete analiza la secuencia de caracteres ingresada y si la sintaxis de la orden es correcta, la ejecuta, recurriendo para ello a las funciones que ofrece el sistema operativo o el programa que representa; la respuesta al usuario se representa en el monitor o en forma de segundo plano. Se trabaja de manera interactiva, es decir, usuario y máquina se comunican de forma sucesiva.

De forma predeterminada, en la mayoría de los shells, el aviso (en inglés, "prompt") consiste en el nombre de la máquina seguido por dos puntos (:), el directorio actual y luego, un carácter que indica el tipo de usuario conectado, como se refleja a continuación:

- "$" define a un usuario normal.
- "#" define al administrador, llamado "root" (12).

Intérpretes de comandos existentes en sistemas UNIX

Uno de los intérpretes más conocidos, es el Shell Bourne, el cual fue el intérprete usado en las primeras versiones de UNIX y se convirtió en un estándar de facto. Al contrario que en el Sistema Operativo de Disco (DOS, por sus siglas en inglés) que el intérprete de comandos es único, en UNIX existen varios. (12)

Estos se mencionan a continuación:

- **Shell Bourne (sh)**: Creado por S. Bourne, es el más utilizado en la actualidad. El prompt del sistema queda representado por el símbolo «$».
- **C-Shell (csh)**: Procedente del sistema BSD, proporciona características tales como control de trabajos, historia de comandos, capacidades de edición, entre otras. Ofrece importantes características para los programadores que trabajan en lenguaje C. Su prompt de sistema queda representado con el símbolo «%».
- **Shell job (jsh)**: Incorpora algunas características de control al shell estándar del sistema.

1.4 Sistemas vinculados al proceso de actualizaciones automáticas en Linux.

Un sistema de gestión de paquetes, también conocido como gestor de paquetes, es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software. El término se usa comúnmente para referirse a los gestores de paquetes en sistemas UNIX, especialmente en Linux, ya que se apoyan considerablemente en estos sistemas de gestión de paquetes.

Herramienta Avanzada de Empaquetado

Herramienta Avanzada de Empaquetado (APT, por sus siglas en inglés), es un sistema de gestión de paquetes creado por el proyecto Debian. APT simplifica en gran medida la instalación y eliminación de programas en los sistemas GNU/Linux. No existe un programa APT en sí, sino una biblioteca de funciones C++ que es empleada por varios programas de línea de comandos para distribuir paquetes, en especial, apt-get y apt-cache. (13)

Existe un repositorio central con más de 25.000 paquetes apt utilizados por apt-get y programas derivados para descargar e instalar aplicaciones directamente desde Internet, conocida como una de las mejores cualidades de Debian. APT fue rápidamente utilizado para funcionar con paquetes .deb, en los sistemas Debian y distribuciones derivadas, pero desde entonces ha sido modificado para funcionar en otros sistemas operativos, como Mac OS X (Fink) y OpenSolaris (distribución Nexenta OS).

Siendo una simple herramienta de línea de órdenes, APT tiene numerosas ventajas frente otras herramientas de gestión de paquetes disponibles para los administradores de sistemas. Algunas de estas ventajas incluyen facilidad de uso a través de conexiones sencillas de terminal (SSH) y la capacidad de poder usarse en scripts de administración del sistema, que pueden automatizarse en la planificación de tareas. Entre sus principales características se encuentran: manejo automático de conflictos, actualización de archivos de configuración para las aplicaciones que así lo requieran y facilidad de uso para usuarios acostumbrados a usar el terminal. (13)

Aptitude

Aptitude es un sistema de gestión de paquetes que proporciona una interfaz para APT. Esta interfaz muestra una lista de paquetes de software y permite al usuario elegir de modo interactivo cuáles desea instalar o eliminar. Dispone de un poderoso sistema de búsqueda que utiliza patrones de búsqueda flexibles, que facilitan al usuario entender las complejas relaciones de dependencia que puedan existir entre los paquetes. Fue diseñado originalmente para distribuciones Debian.

Aptitude se basa en una biblioteca ncurses, mediante la cual provee una interfaz que incorpora algunos elementos comunes a otras interfaces gráficas, como son los menús desplegables. Para utilizar Aptitude por línea de comandos, al igual que APT, se debe tener permisos de superusuario, ya que estas herramientas pueden realizar cambios significativos en el sistema.

Aptitude se creó para experimentar con un diseño más orientado a objetos que el usado en console-apt, con la esperanza de que resultara un programa mucho más flexible y con un conjunto de características extensibles. Actualmente está considerado una de las mejores alternativas basadas en terminal. (14)

Ventajas de Aptitude frente a APT: (13)

- Sigue la pista de todos los paquetes instalados. Cuando es usado en todas las instalaciones, recordará aquellos que sólo se necesitaban para satisfacer dependencias y los borrará cuando ya no sean necesarios.
- Maneja librerías recomendadas, apt-get se limita a informar de la recomendación.
- La interfaz de Aptitude es muy eficaz y tiene gran capacidad para búsquedas. Se pueden visualizar paquetes por categorías, buscar y filtrar paquetes por nombre, descripción, responsable y dependencias.
- Muestra en una sección paquetes creados por el usuario, que no son parte de la distribución.
- Admite usar diversas fuentes, permitiendo especificar la versión del paquete a instalar.
- Guarda un log de sus acciones.

Synaptic

Synaptic es un gestor de paquetes que provee una interfaz gráfica para APT. Combina la simplicidad de la interfaz gráfica de usuario con la potencia de la herramienta de línea de comandos «apt». Permite instalar, eliminar, configurar y actualizar los paquetes de software, explorar por tipos de aplicaciones y buscar la lista de paquetes de software disponibles; gestionar los repositorios y actualizar el sistema completamente. Además de poner en cola varias acciones antes de ejecutarlas; informa sobre las dependencias (paquetes adicionales requeridos por el paquete que se haya escogido), así como, los posibles conflictos con otros paquetes que ya estén instalados en el sistema. (15)

Mantiene una base de datos de paquetes del sistema con el objetivo de tener un seguimiento del software instalado. Esta lista se chequea y se compara con la lista de repositorios para informar de nuevos paquetes o actualizaciones, comprueba si hay nuevos paquetes cada vez que se inicia.

Synaptic es una aplicación mucho más avanzada que el Centro de software de Ubuntu, es un gestor de paquetes muy potente. Ofrece mucha más información a través de su interfaz gráfica y asegura un completo control sobre la gestión de los paquetes.

Synaptic proporciona las características siguientes: (16)

- Buscar paquetes por nombre, descripción y otros atributos.
- Seleccionar paquetes por estado, sección, nombre o un filtro personalizado.
- Ordenar paquetes por nombre, estado, tamaño o versión.
- Examinar toda la documentación disponible en línea relacionada con un paquete.

1.5 Metodología, herramientas y lenguajes.

1.5.1 Metodología.

Proceso Unificado de Desarrollo (RUP)

El Proceso Unificado de Desarrollo (RUP, por sus siglas en inglés), es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado (UML), constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. (17)

RUP es el resultado de varios años de desarrollo y uso práctico, en el que se han unificado técnicas de desarrollo y trabajo de un conjunto de metodologías, adaptables al contexto y necesidades de cada organización. No existen dos proyectos de desarrollo de software que sean iguales. Cada uno tiene prioridades, requerimientos, y tecnologías muy diferentes. Sin embargo, en todos los proyectos, se debe minimizar el riesgo, garantizar la predictibilidad de los resultados y entregar software de calidad superior a tiempo. RUP es una plataforma flexible de procesos de desarrollo de software que ayuda suministrando guías consistentes y personalizadas de procesos, para todo el equipo del proyecto.

Una de sus mejores prácticas centrales es la noción de desarrollar iterativamente. Organiza los proyectos en términos de disciplinas y fases, consistiendo cada una, en una o más iteraciones. Con esta aproximación iterativa, el énfasis de cada flujo de trabajo variará a través del ciclo de vida. La aproximación iterativa ayuda a mitigar los riesgos en forma temprana y continua, con un progreso demostrable. (18)

Características principales de la metodología RUP:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Desarrollo iterativo e incremental.
- Centrado en la arquitectura y guiado por los casos de uso.
- Administración de requisitos.
- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.

1.5.2 Herramientas.

Enterprise Architect 7.1

Enterprise Architect es una potente herramienta de análisis y diseño para el desarrollo de software robusto y mantenible; desde la recogida de requisitos, pasando por el análisis, modelado, implementación y pruebas, hasta despliegue y mantenimiento. Es una herramienta de modelado potente, versátil y multiusuario, que ayuda a que los proyectos de desarrollo de software tengan éxito. Soporta generación de ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Java, Delphi, VB.Net, Visual Basic y PHP. (19)

Enterprise Architect combina el poder de la última especificación UML 2.1 con alto rendimiento e interfaz intuitiva, para traer modelado avanzado al escritorio y para el equipo completo de desarrollo e implementación. Cuenta con un entorno gráfico en el cual se pueden construir los diagramas y producir imágenes fáciles de ver. Además tiene diversas características avanzadas que mejoran el uso de UML, como son: la capacidad de generar documentación definible en formatos RTF y HTML y exportar código en una gran variedad de lenguajes.

Algunas de las características claves de Enterprise Architect son:

- ✓ Alta capacidad.
- ✓ Velocidad, estabilidad y buen rendimiento.

- ✓ Trazabilidad de extremo a extremo.
- ✓ Construido sobre las bases de UML 2.1.
- ✓ Soporte para los 13 diagramas de UML 2.

Visual Studio 2005

Visual Studio 2005 es una completa herramienta de desarrollo de software, trabaja con Framework 2.0; permite crear e implementar rápidamente una amplia gama de aplicaciones de Windows, web y dispositivos móviles. Es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) que soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. (20)

Cuenta con un entorno de desarrollo altamente productivo con diseñadores visuales, lenguajes de programación y editores de código mejorados. Desarrolla y depura aplicaciones multicapa de servidor desde un mismo entorno unificado de desarrollo. Permite construir soluciones para SQL Server 2005 utilizando herramientas visuales integradas de diseño de bases de datos e informes. Además 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 gran variedad de plataformas.

Framework Microsoft.NET 2.0

El Framework Microsoft.NET 2.0 es un componente que pueden incluir los sistemas operativos Windows, para el desarrollo de aplicaciones y Servicios Web XML. Está diseñado con el fin de cumplir los siguientes objetivos:

- Proporcionar un entorno de ejecución de código que minimice los conflictos de despliegue y versionado de software; promueva la ejecución segura del código, incluyendo código creado por terceros desconocidos y elimine los problemas de rendimiento de los entornos para la interpretación de código.
- Desarrollar estándares de comunicación, para garantizar que el código basado en el Framework.NET se pueda integrar con cualquier otro tipo de código.

- Hacer que la experiencia de los desarrolladores sea más agradable a través de la implementación de los distintos tipos de aplicaciones, como las de escritorio o para la Web.

El Framework Microsoft.NET tiene dos componentes principales: el Lenguaje Común en Tiempo de Ejecución (CLR, por sus siglas en inglés) y la librería de clases base. El CLR provee lo que se llama código administrado, es decir, un entorno que brinda servicios automáticos al código que se ejecuta, tales como: gestión de memoria, manejo de hilos e interacción remota, al tiempo que ofrece comprobación estricta de tipos y otras precisiones del código que promueven la seguridad y robustez de las aplicaciones. (21)

La librería de clases base es el otro componente principal del Framework.NET, sobre ésta se construyen todas las demás clases que utilizan los programas de Visual Studio .NET. Esta librería es independiente del lenguaje, permite el uso y la depuración de otros lenguajes, y es extensible. (21)

MonoDevelop 2.0

MonoDevelop es un entorno de desarrollo libre y gratuito creado por desarrolladores provenientes del Proyecto Mono, quienes basándose en el popular IDE SharpDevelop para Windows lo adaptaron al entorno Linux usando las librerías Gtk. (22)

Es un IDE bastante flexible pensado originalmente para desarrollar aplicaciones de software con C# y otros lenguajes de programación como C/C++, Visual Basic .NET, Java, Vala y Boo. Lo novedoso de este entorno de desarrollo es que permite crear de manera rápida aplicaciones ASP.NET sobre Linux, permitiendo además migrar aplicaciones .NET de Microsoft Visual Studio a Linux usando el mismo código fuente.

Entre sus principales características se encuentran: (23)

- **Edición avanzada de texto:** Soporte para autocompletado de código y sintaxis para cualquiera de los lenguajes compatibles con el IDE.
- **Depurador Integrado:** Nuevo depurador para aplicaciones ASP.NET y nativas.

- **Entorno y herramientas visuales configurables:** De forma similar a NetBeans, MonoDevelop permite arrastrar y reordenar los elementos visuales y cajas de herramientas según las preferencias del usuario.
- **Diseñador Visual para GTK+:** Formularios para entornos visuales utilizando el motor Gtk+.
- **Herramientas avanzadas:** Control de código fuente, integración de Makefiles, pruebas unitarias y empaquetado de aplicaciones.

Framework Mono 2.0

Mono es la implementación libre de la Infraestructura de Lenguaje Común (CLI, por sus siglas en inglés) y C#, de acuerdo a las especificaciones enviadas a la Asociación Europea de Fabricantes de Ordenadores (ECMA, por sus siglas en inglés) para su estandarización. Esta implementación es de código fuente abierto (Open Source).Mono también incluye un compilador de C#, el cual paradójicamente esta escrito en C# al igual que el CLI. (24)

Es una herramienta para el desarrollo de código abierto, basada en el Framework Microsoft.NET que permite a los desarrolladores construir aplicaciones para Linux y multiplataformas. Brinda soporte para lenguajes como: C#, Visual Basic, Java, Python, Ruby y F#.

Adicionalmente, Mono cuenta con un catálogo de librerías compatibles con las librerías del Framework Microsoft.NET, pero además, cuenta con una serie de librerías no existentes en éste; como el GTK# que permite crear interfaces gráficas nativas del toolkit GTK+, Mono.LDAP y Mono.Posix. (25)

La motivación de crear Mono se debe a la búsqueda de herramientas, que ayudaran a la creación rápida de aplicaciones en el entorno Linux.

Plataformas soportadas por Mono:

Actualmente Mono se puede ejecutar en plataformas x86, PPC, SPARC y S390 en 32 bits; y x86-64 y SPARC en 64 bits; siendo posible crear y ejecutar aplicaciones en los sistemas operativos: GNU\Linux, FreeBSD, UNIX, Mac OS X, Solaris y Windows.

Requerimientos para que una aplicación sea compatible con Mono y el Framework.Net: (25)

- Linux distingue entre mayúsculas y minúsculas en los nombres de archivos y directorios; por lo que es necesario tener una consistencia con los nombres que se utilicen.
- El separador de rutas es diferente en Windows (\) que en Linux (/), por lo tanto se recomienda usar el API Path.DirectoryPathSeparator, para obtener en separador correcto al ejecutar la aplicación.
- Si se utilizan librerías no CLI (Ej: librerías en C, C++ etc.), mediante p/Invoke, asegurarse que la librería existe en los diferentes ambientes donde se va a ejecutar la aplicación.
- No hacer uso de tecnologías que solo existan en un ambiente en particular (Ej: Registry en Windows o GConf en Linux -Gnome-); o bien proveer alguna solución que permita a la aplicación operar de forma correcta en el ambiente donde se ejecuta.
- Aplicaciones basadas en Windows Forms y que sean muy complejas pueden no funcionar de momento, debido a que Windows Forms en Mono no esta completo.

NAnt

NAnt es una herramienta de código abierto para automatizar procesos. Constituye una adaptación de Apache Ant para .NET. Es similar a ANT, pero para los ambientes .NET, en vez de Java. Se puede extender mediante clases Tasks (tareas), no mediante Shell (comandos), como la mayoría de estas herramientas; esta característica la hace multiplataforma.

Los archivos de configuración se basan en sintaxis XML para ejecutar una secuencia de tareas definidas para compilar aplicaciones .NET, tienen extensión .build y están compuestos por cuatro tipos de ítems (etiquetas): Tasks (tareas), Targets (nodos), Properties (propiedades) y Projects (proyectos). En los nodos Targets 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. (26)

Soporta la compilación de archivos:

- C#
- VB.NET
- C++
- J#

Entre las tareas más importantes que implementa la más reciente versión de NAnt se encuentran:

- <Copy> Copia un archivo o conjunto de archivos a un nuevo archivo o directorio.
- <Delete> Elimina un solo archivo o todos los archivos en un directorio especificado y sus subdirectorios.
- <Move> Mueve un archivo o conjunto de archivos a un nuevo archivo o directorio.
- <Zip> Crea un archivo zip de los conjuntos de archivos especificados.
- <Tar> Crea un archivo tar del conjunto de archivos especificados.

De esta herramienta sólo se utiliza su esquema para dar formato al archivo XML.

1.5.3 Lenguajes.

Lenguaje de Marcas Extensible (XML)

Lenguaje de Marcas Extensible (XML, por sus siglas en inglés). Permite definir la gramática de lenguajes específicos, por lo tanto, XML no es realmente un lenguaje en particular; sino una manera de definir lenguajes para diferentes necesidades. Impone una sintaxis más rígida para las marcas que permite su proceso de forma más eficiente, también distingue entre minúsculas y mayúsculas. (27)

XML permite definir lenguajes de marcas para cualquier fin y tiene capacidades de validación de datos. Algunos de estos lenguajes que usan XML para su definición son: Lenguaje Extensible de Marcado de Hipertexto (XHTML, por sus siglas en inglés) y Gráficos Vectoriales Escalables (SVG, por sus siglas en inglés).

Es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con posibilidades 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.

Características principales de XML: (28)

- Proporciona diferentes vistas sobre los datos (HTML, PDF, entre otros), dependiendo de quién sea el cliente.
- Facilita la integración desde fuentes de datos heterogéneas, por ejemplo: páginas Web y distintas bases de datos.
- Los documentos tienen una estructura que los hace legibles no sólo para los ordenadores, si no también para los humanos.
- Las aplicaciones de XML son fácilmente extensibles mediante definiciones de nuevos tipos de documento (DTD).

Lenguaje Unificado de Modelado (UML 2.1)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). Para comprender qué es el UML, basta con analizar cada una de las palabras que lo componen, por separado. (29)

- Lenguaje: el UML es, precisamente, un lenguaje. Lo que implica que éste cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.
- Modelado: el UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.
- Unificado: unifica varias técnicas de modelado en una única.

Debido a que el UML proviene de técnicas orientadas a objetos, se crea con la fuerte intención de que éste permita un correcto modelado orientado a objetos. Con su versión más reciente, UML 2.1, el estándar para la definición de sistemas de software se ha extendido a la dirección funcional, incluye la aplicación de redefinición y asociaciones bidireccionales en el metamodelo. UML 2.1 proporciona un conjunto de Gráficos Modeling Framework (GMF) que son editores basados en el metamodelo UML2 MDT.

Lenguaje de programación CSharp(C#)

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft Corporation como parte de su iniciativa .NET en respuesta al éxito del lenguaje Java de Sun Microsystems. El código fuente de C#, así como el de otros lenguajes .NET, es compilado en un código intermedio llamado Lenguaje Intermedio de Microsoft (MSIL, por sus siglas en inglés). C# deriva principalmente de los lenguajes C, C++ y Java con algunas características de Visual Basic de Microsoft. (30)

Es usado para desarrollar aplicaciones para el entorno .NET y ofrece una alternativa al desarrollo basado en Java. El entorno de desarrollo Visual Studio de Microsoft incorpora varios lenguajes diferentes incluyendo ASP.NET, C#, C++ y J# (Microsoft Java para .NET), los cuales compilan al Lenguaje común en tiempo de ejecución (CLR, por sus siglas en inglés).

Principales características del lenguaje C#: (31)

- **Sencillez:** Elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. El código escrito en C# es autocontenido, lo que significa que no necesita de ficheros adicionales.
- **Modernidad:** Incorpora en el propio lenguaje, elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes, como Java o C++ hay que simular.
- **Orientado a objetos:** Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos. Una diferencia de este enfoque orientado a objetos, respecto al de otros lenguajes como C++, es que el de este lenguaje es más puro. Esto se debe a que no admite ni funciones ni variables globales; sino que todo el código y datos han de especificarse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.
- **Seguridad de tipos:** Incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto. Esto es especialmente necesario en un entorno gestionado por un recolector de basura.

- **Versionable:** Incluye una política de versionado que permite crear nuevas versiones de tipos, sin temor a que la introducción de nuevos miembros, provoquen errores difíciles de detectar en tipos hijos previamente desarrollados y ya extendidos, con miembros de igual nombre a los recién introducidos.

1.6 Estilo arquitectónico.

Un estilo arquitectónico describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción. Estos guían a la organización del sistema de software e incluyen reglas y líneas a seguir para la organización de un sistema. (32)

Patrón de diseño Modelo-Vista-Controlador (MVC)

El patrón de diseño MVC se utiliza para separar la información, la salida y el procesamiento de los datos de la aplicación. La aplicación se divide en tres capas: el modelo, la vista y el controlador; cada elemento gestiona una parte distinta del proceso.

El modelo: Incorpora los datos y las reglas de la aplicación. Gran parte del procesamiento de la aplicación tiene lugar en esta parte del patrón de diseño. No se aplica ningún formato a los datos devueltos para la interfaz (o procesador principal) de la aplicación en esta parte del proceso. Los datos devueltos se pueden utilizar para distintas interfaces (o vistas).

La vista: Gestiona la interfaz con la que interactúa el usuario y representa el contenido del modelo. La interfaz especifica la forma en que se presentan los datos del modelo y permite al usuario acceder a los datos de la aplicación.

El controlador: Suele contener mucho código. Realiza llamadas a cualquier parte del modelo, según las peticiones realizadas por el usuario desde la vista y contiene código específico de la aplicación. Puesto que este código es específico de la aplicación, no suele ser reutilizable. Sin embargo, el resto de componentes del patrón de diseño sí se pueden reutilizar. El controlador no procesa ni produce ningún dato: recibe la petición del usuario y decide a qué parte del modelo o componentes de la vista necesita

llamar, determina dónde se envían los datos y qué formato se aplica a los datos devueltos. Normalmente, el controlador transmite y responde ante cambios que afectan al modelo y a la vista. (33)

Ventajas del uso del patrón de diseño MVC:

- Clara separación entre el modelo, la vista y el controlador.
- Sencillez para crear distintas representaciones de los mismos datos.
- Facilidad para la realización de pruebas unitarias de los componentes.
- Reutilización de los componentes.
- Simplicidad en el mantenimiento de los sistemas.
- Facilidad para desarrollar prototipos rápidos.
- Los desarrollos suelen ser más escalables.

Teniendo en cuenta lo investigado para el desarrollo de este capítulo, se puede concluir que para llevar a cabo la construcción de un intérprete de comandos, es necesario tener en cuenta varios puntos y factores indispensables para que sea desarrollado con la calidad requerida. Se sugiere el empleo de las herramientas descritas anteriormente, para la obtención del sistema. Las mismas poseen un ambiente de integración propicio para el desarrollo de la solución propuesta.

Capítulo 2. Características del sistema.

Con el desarrollo del presente capítulo se tiene como objetivo realizar una descripción detallada de las características del sistema para lograr un mejor entendimiento del mismo. Debido a las especificidades de éste, es necesaria la realización de un Modelo de Dominio, en el que se precisan conceptos asociados al sistema. Además se exponen los requisitos funcionales y no funcionales a partir de los cuales se obtendrán los casos de uso.

2.1 Modelo de dominio.

El Modelo de dominio es una representación visual estática del entorno real del proyecto. Es decir, un diagrama con los objetos que existen asociados al proyecto que se va a acometer y las relaciones que hay entre ellos. Se debe tener en cuenta que no son clases de software, aunque algunos objetos del Modelo de dominio pueden terminar siéndolo. En este contexto, el término “dominio” representa una parte del negocio. El Modelo de dominio ayuda a comprender los conceptos que utilizan los usuarios, con los que trabajan y con los que deberá trabajar la aplicación.

2.1.1 Conceptos del Modelo de dominio.

Técnico

Persona encargada de iniciar y asistir el flujo de eventos de una actualización automática.

Aplicación

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

Paquete de Actualización

Contiene todos los componentes necesarios dígase (.xml, .exe, .html, .cs entre otros) para dar comienzo al flujo de actividades de una actualización.

Manifiesto

La clase manifiesto es un documento XML que está contenido en el paquete de actualización. Posee un listado con 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 pueden 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, el cual es almacenado en una dirección local de la PC.

Recurso de Versiones

Una clase recurso 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.

2.1.2 Diagrama de clases del Modelo de Dominio.

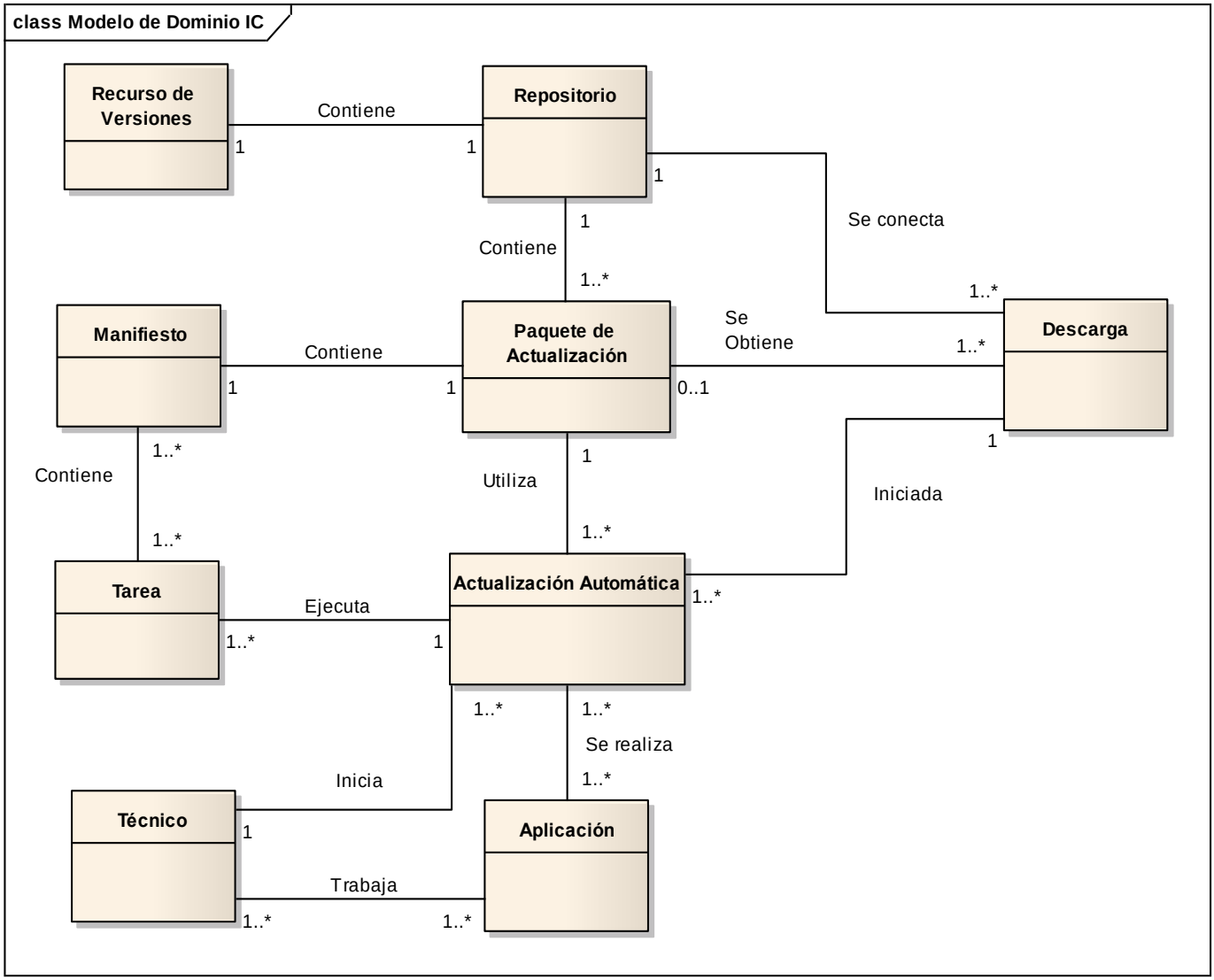

Fig.2 Diagrama de clases del Modelo de Dominio.

2.2 Especificación de requisitos.

Los requisitos o requerimientos son una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente.

2.2.1 Requisitos Funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, describen la funcionalidad o los servicios que se espera que éste provea. Estos dependen del tipo de software, del sistema que se desarrolle y de los posibles usuarios del software.

RF1. Configurar Aplicación

RF2. Modificar configuración de aplicación

RF3. Actualizar Aplicaciones

RF4. Mostrar Actualizaciones Disponibles

RF5. Descargar Recurso de Versiones

RF6. Descargar Paquete de Actualización

RF7. Escanear Paquete de Actualización

RF8. Realizar salva de aplicación

RF9. Deshacer actualización

RF10. Restablecer actualización

RF11. Mostrar actualizaciones a restablecer

RF12. Mostrar información de actualizaciones realizadas

RF13. Buscar información de actualizaciones realizadas

RF14. Eliminar archivos temporales

2.2.2 Requisitos no Funcionales.

Los requisitos no funcionales son propiedades o cualidades que el sistema debe poseer. Forman una parte significativa de la especificación y son importantes para que clientes y usuarios, puedan valorar las características no funcionales de la aplicación desarrollada. Si se conoce, que un sistema cumple con las funcionalidades requeridas; las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable es éste, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación.

Usabilidad

RNF1. El sistema podrá ser usado por los usuarios que sean administradores en el equipo local y que a su vez, posean una cuenta de administración en la entidad a la que pertenecen. Estos serán los encargados de interactuar con la aplicación especificando las acciones que desean llevar a cabo.

RNF2. Las estaciones de trabajo donde se desplegará el sistema deben tener instalado el Framework Microsoft.NET o Framework Mono en dependencia del sistema operativo que se esté utilizando, ambos en su versión 2.0 o superior.

Fiabilidad

RNF3. Se mantendrá seguridad y control a nivel de usuarios de sesión. Si el usuario autenticado en el sistema operativo no es un administrador del equipo se le negará el acceso al sistema.

RNF4. El sistema proveerá al administrador que esté llevando a cabo el proceso de instalación, de una contraseña como root del sistema y la posibilidad de gestionar los permisos de usuario consumiendo un servicio WEB.

RNF5. El sistema permitirá el restablecimiento de la configuración anterior en caso de errores durante la actualización y restablecimiento de versiones, a partir de los respaldos o salvas realizadas de cada uno de los ficheros modificados.

RNF6. El Actualizador Automático podrá ser utilizado en todo momento.

Eficiencia

RNF7. El Actualizador Automático deberá ser rápido ante la solicitud de llevar a cabo alguna opción que éste brinda.

Soporte

RNF8. Se permitirá realizar copias de seguridad de los archivos guardados en las salvas hacia otro dispositivo de almacenamiento externo, además de recuperar los mismos a partir de los respaldos realizados.

RNF9. Una vez terminada la extensión del Actualizador Automático se realizarán las pruebas del mismo.

Restricciones del diseño

RNF10. Para el desarrollo se utilizará C# como leguaje de programación.

RNF11. Se utilizará MonoDevelop 2.0 y Visual Studio 2005 como entornos de desarrollo.

RNF12. Para realizar el modelado del software se utilizará la herramienta Enterprise Architect 7.1.

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

RNF13. Se dispondrá de la documentación del sistema realizada con la metodología de desarrollo RUP y de un Manual de Usuario que indicará los pasos necesarios para interactuar con las funcionalidades del Actualizador Automático.

Interfaces Hardware

RNF14. Requerimientos mínimos:

Procesador	Memoria RAM	Espacio en disco	
		SO_ 32 bit	SO_64 bit
400 MHz	96 MB	280 MB	610 MB

Tabla 2.1 Requerimientos mínimos de hardware.

Estándares Aplicables

RNF15. Se utilizará RUP como metodología de desarrollo de software y el lenguaje UML 2.1 para visualizar, especificar, construir y documentar los artefactos del sistema.

2.3 Modelado del sistema.

2.3.1 Actores del sistema.

Actor	Descripción
Administrador	Encargado de interactuar con la aplicación especificando las acciones que desea llevar a cabo.

2.3.2 Diagrama de Casos de Uso del sistema.

Los diagramas de casos de usos son representaciones gráficas en las que se muestran los requisitos funcionales que se esperan de un sistema y su relación con el entorno. Los casos de uso son utilizados básicamente en el proceso de modelado de sistemas. Ver (Fig.3).

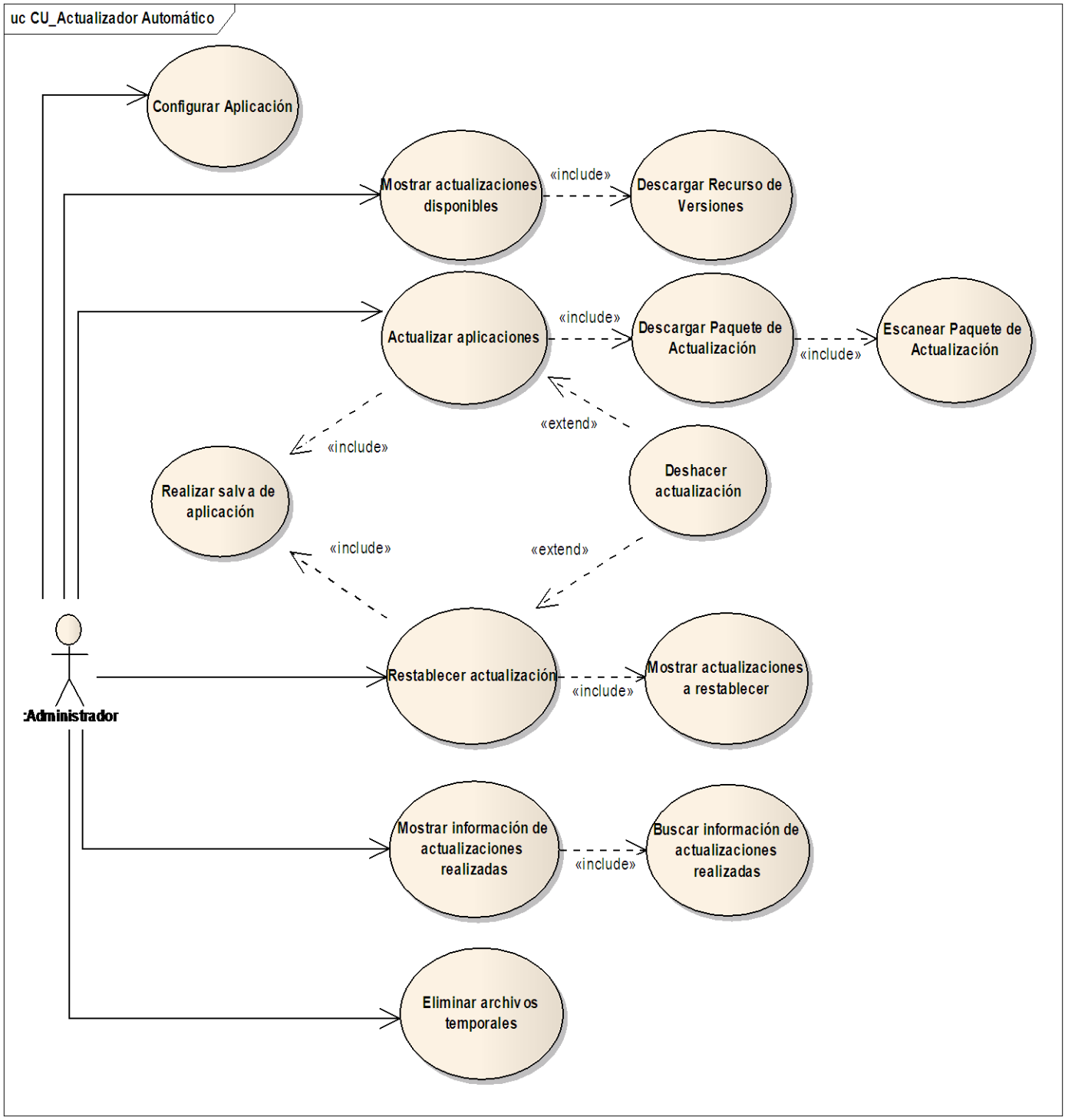

Fig.3 Diagrama de Casos de Uso del sistema.

2.3.3 Especificación de los casos de uso.

Descripción del Caso de Uso: Configurar Aplicación

Caso de Uso:	Configurar Aplicación.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para configurar la aplicación. Éste modifica o crea una nueva configuración, el sistema guarda la configuración y el caso de uso termina.
Precondiciones:	Que el usuario tenga permisos de ejecución.
Referencias	RF1,RF2
Prioridad	**Crítico**

Descripción del Caso de Uso: Actualizar aplicaciones

Caso de Uso:	Actualizar aplicaciones.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para actualizar las aplicaciones seleccionadas, el sistema lleva a cabo la actualización y el caso de uso termina.
Precondiciones:	Que existan actualizaciones disponibles.
Referencias	RF3
Prioridad	**Crítico**

Descripción del Caso de Uso: Deshacer actualización

Caso de Uso:	Deshacer actualización.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando ocurre un error durante una actualización o un restablecimiento de una actualización, el sistema devuelve la aplicación a la versión en la que se encontraba al dar inicio la operación y termina el caso de uso.
Precondiciones:	Que existan las salvas de seguridad de la aplicación.
Referencias	RF9
Prioridad	**Crítico**

Descripción del Caso de Uso: Restablecer actualización

Caso de Uso:	Restablecer actualización.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para restablecer una actualización, el sistema comienza a deshacer cada una de las tareas que fueron ejecutadas durante la actualización. Una vez que termine el proceso se le informa al actor y el caso de uso termina.
Precondiciones:	Que existan las copias de seguridad de la aplicación.
Referencias	RF10
Prioridad	**Crítico**

En este capítulo se han dado a conocer y explicado las clases del Modelo de Dominio relacionadas con el sistema ,así como, los requisitos funcionales a partir de los cuales se obtuvieron los casos de uso y los requisitos no funcionales que debe cumplir el mismo. Lo anterior expuesto posibilita una mejor comprensión del sistema que se desarrollará y permitirá comenzar con el análisis y diseño del sistema propuesto.

Capítulo 3. Análisis y diseño del sistema.

El presente capítulo tiene como objetivo la confección de los diagramas de clases del análisis, los de clases del diseño y los diagramas de interacción; aplicando distintas técnicas y principios con el propósito de definir un producto con suficientes detalles que permitan su desarrollo. Se realizará además la descripción de la arquitectura, siendo estos los artefactos principales que contribuyen a la implementación del sistema.

3.1 Modelo de análisis.

Durante el análisis se estudian los requisitos que fueron descritos en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción que sea fácil de entender y que ayude a estructurar todo el sistema, incluyendo su arquitectura.

3.1.1 Diagramas de clases del análisis.

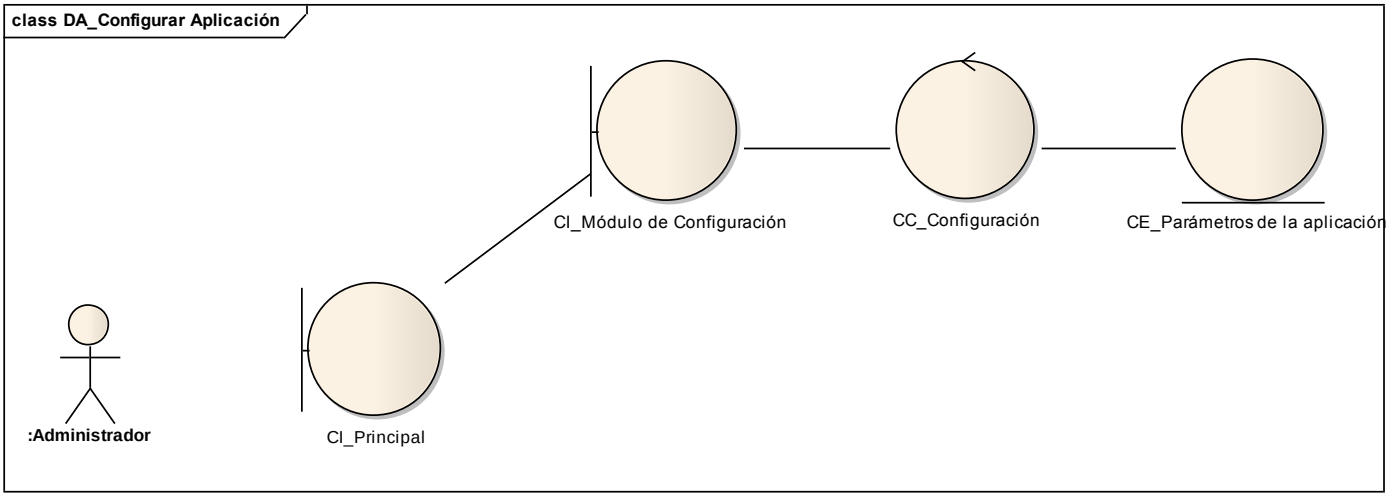

Fig.4 Diagrama de clases del análisis del CU Configurar Aplicación.

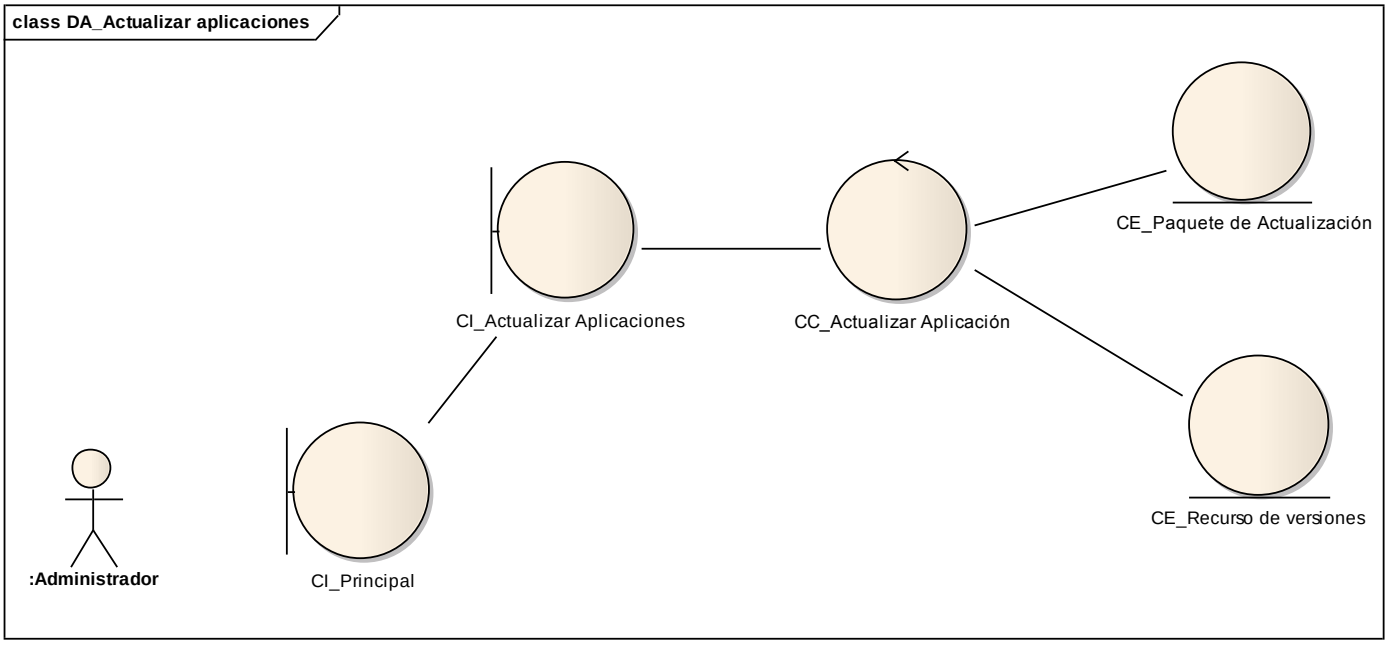

Fig.5 Diagrama de clases del análisis del CU Actualizar aplicaciones.

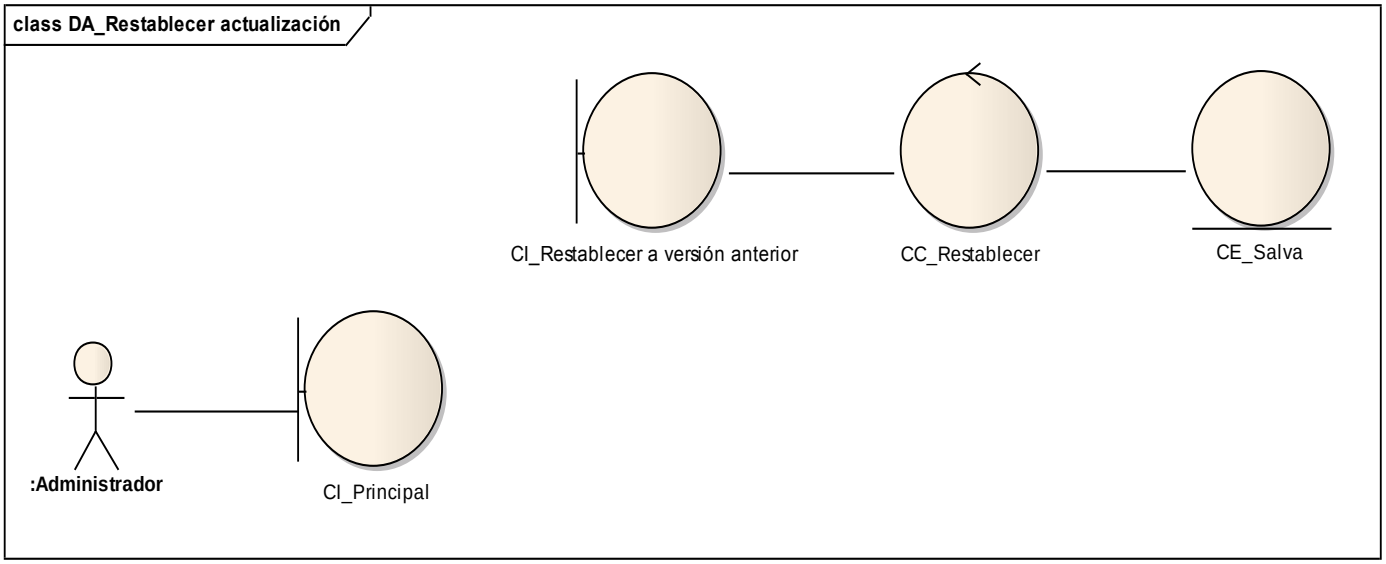

Fig.6 Diagrama de clases del análisis del CU Restablecer actualización.

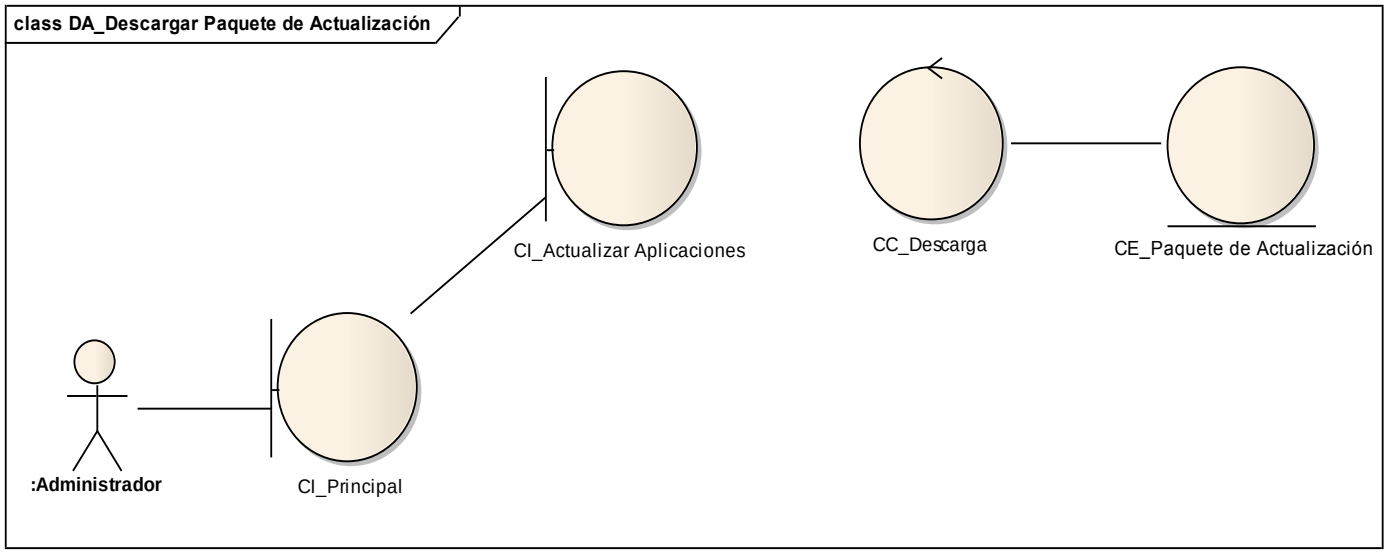

Fig.7 Diagrama de clases del análisis del CU Descargar Paquete de Actualización.

3.2 Arquitectura del sistema.

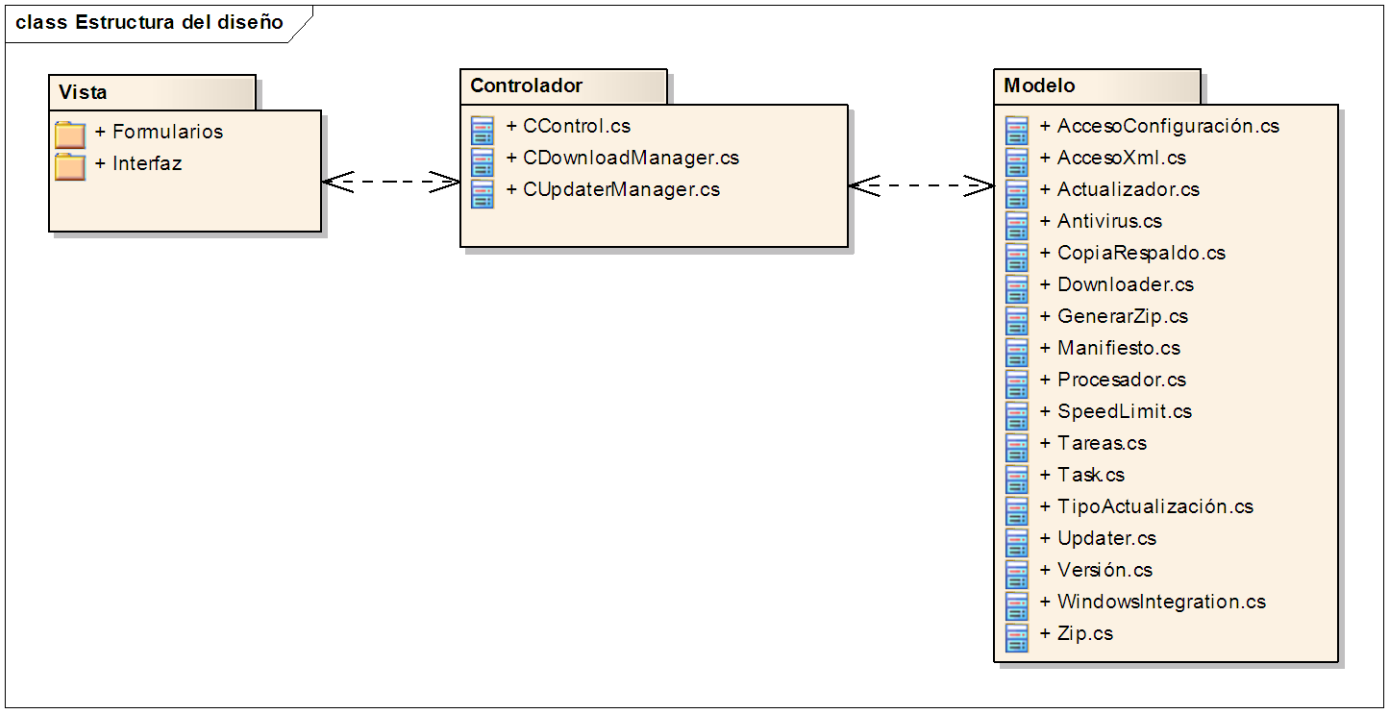

Fig.8 Arquitectura del sistema.

La Arquitectura de Software es, a grandes rasgos, una vista del sistema que incluye los componentes principales, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. Para la solución propuesta se utilizará el patrón de diseño Modelo-Vista-Controlador, las tres capas en las que se divide este patrón son las siguientes:

Vista: Permite la interacción del usuario con el sistema. Captura y comunica la información del usuario, ejecutando el mínimo de procesos y realizando un filtrado previo, para comprobar que no hay errores de formato. Se comunica únicamente con el Controlador. Esta integrada básicamente por las interfaces y controles de usuario.

Controlador: Es el encargado de recibir las peticiones de los usuarios e iniciar correctamente los procesos pertinentes residentes en el Modelo, así como, enviar las respuestas obtenidas de esa interacción a la Vista. Esta integrada básicamente por clases controladoras que administran objetos y entidades fuertemente relacionadas.

Modelo: Está integrado por los objetos de acceso a datos, la abstracción de los datos y otros objetos que agrupan funcionalidades complementarias, en fin la lógica del negocio. Se comunica únicamente con el Controlador. Es el encargado de almacenar, adicionar y modificar los datos del negocio, así como, ejecutar los procesos necesarios para responder a las peticiones realizadas por el Controlador y que son resultado de la interacción con el usuario.

3.2.1 Patrones de diseño que se utilizarán durante el desarrollo del sistema.

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 este ambiente, y luego describe el núcleo de la solución a ese problema. Estos patrones se han convertido en la metodología por excelencia de diseño y posterior 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.

Patrones de Asignación de Responsabilidades (GRASP)

Los Patrones Generales de Software para Asignar Responsabilidades (GRASP, por sus siglas en inglés), describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

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 basa 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. (34)

A continuación se nombran los patrones básicos de asignación de responsabilidades:

- Patrón Experto
- Patrón Creador
- Patrón Bajo Acoplamiento
- Patrón Alta Cohesión
- Patrón Controlador

Patrones de diseño en la Programación Orientada a Objetos

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. Los patrones de diseño no son fáciles de entender, pero una vez entendido su funcionamiento, los diseños serán mucho más flexibles, modulares y reutilizables. Han revolucionado el diseño orientado a objetos y todo buen arquitecto de software debería conocerlos. A continuación se darán a conocer algunos que se utilizarán en el desarrollo del sistema. (35)

Decorator

El patrón Decorator responde a la necesidad de añadir dinámicamente funcionalidad a un objeto. Esto evita tener que crear sucesivas clases que hereden de la primera incorporando la nueva funcionalidad, sino otras que la implementan y se asocian a la primera.

Builder

Como Patrón de diseño, el patrón Builder (Constructor) es usado para permitir la creación de una variedad de objetos complejos desde un objeto fuente (Producto), el objeto fuente se compone de una variedad de partes que contribuyen individualmente a la creación de cada objeto complejo a través de un conjunto de llamadas a interfaces comunes de la clase Abstract Builder.

Singleton

El patrón de diseño Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su objetivo es garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

3.3 Modelo de diseño.

En el diseño se modela el sistema para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones de estos. Una entrada esencial en el diseño es el resultado del análisis, o sea el Modelo de análisis, que proporciona una comprensión detallada de los requisitos. Además impone una estructura del sistema que se debe conservar lo más fielmente posible cuando se de forma al sistema.

Principales propósitos del diseño:

- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables.

3.3.1 Diagramas de clases del diseño.

Un diagrama de clases del diseño representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Éste da forma al sistema mientras que intenta preservar la estructura definida en el modelo de análisis. No todas las clases que aparecían en el Modelo Conceptual tienen por qué aparecer en el diagrama de clases del diseño; solo se incluirán aquellas clases a las que se les ha asignado algún tipo de responsabilidad en el diseño del sistema.

3.3.2 Diagramas de secuencia.

Los diagramas de secuencia de un sistema muestran gráficamente los eventos que originan los actores y que tienen un impacto en el sistema. Además describen las interacciones entre los actores y el sistema mostrando de forma secuencial los envíos de mensajes entre ellos. A continuación se muestran los principales diagramas de secuencia por casos de uso del sistema. Ver (Fig.9 y 10).

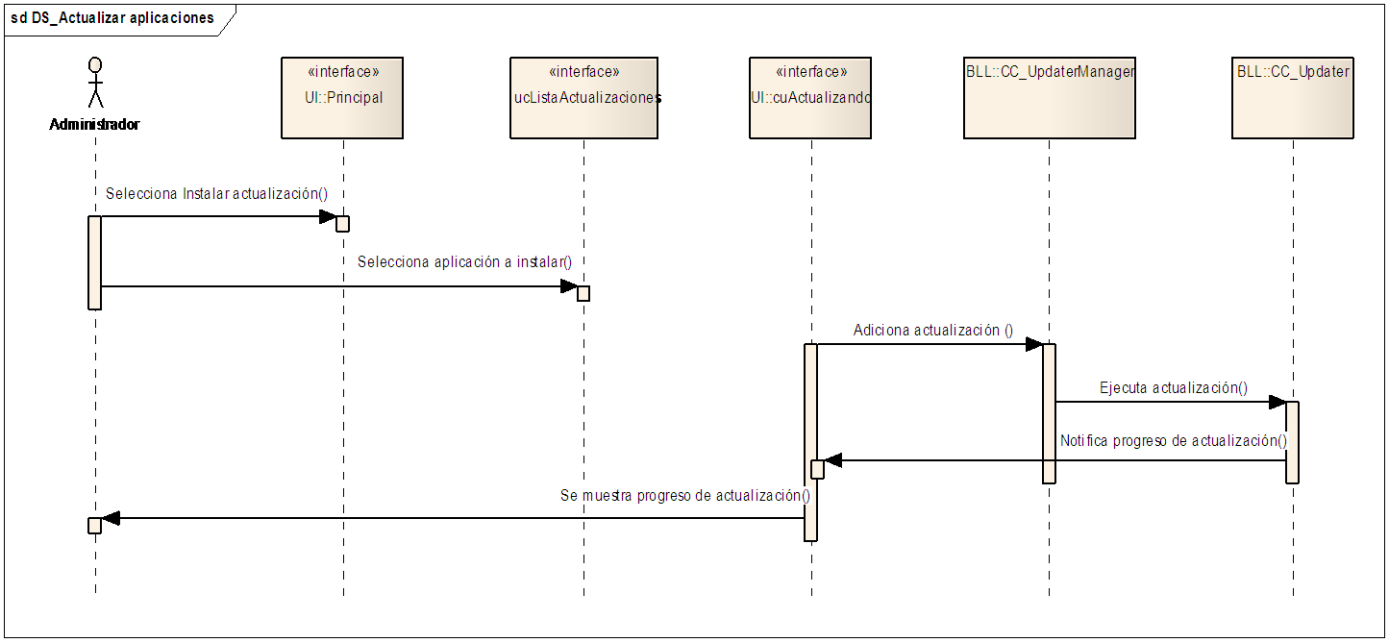

Fig.9 Diagrama de secuencia del CU Actualizar aplicaciones.

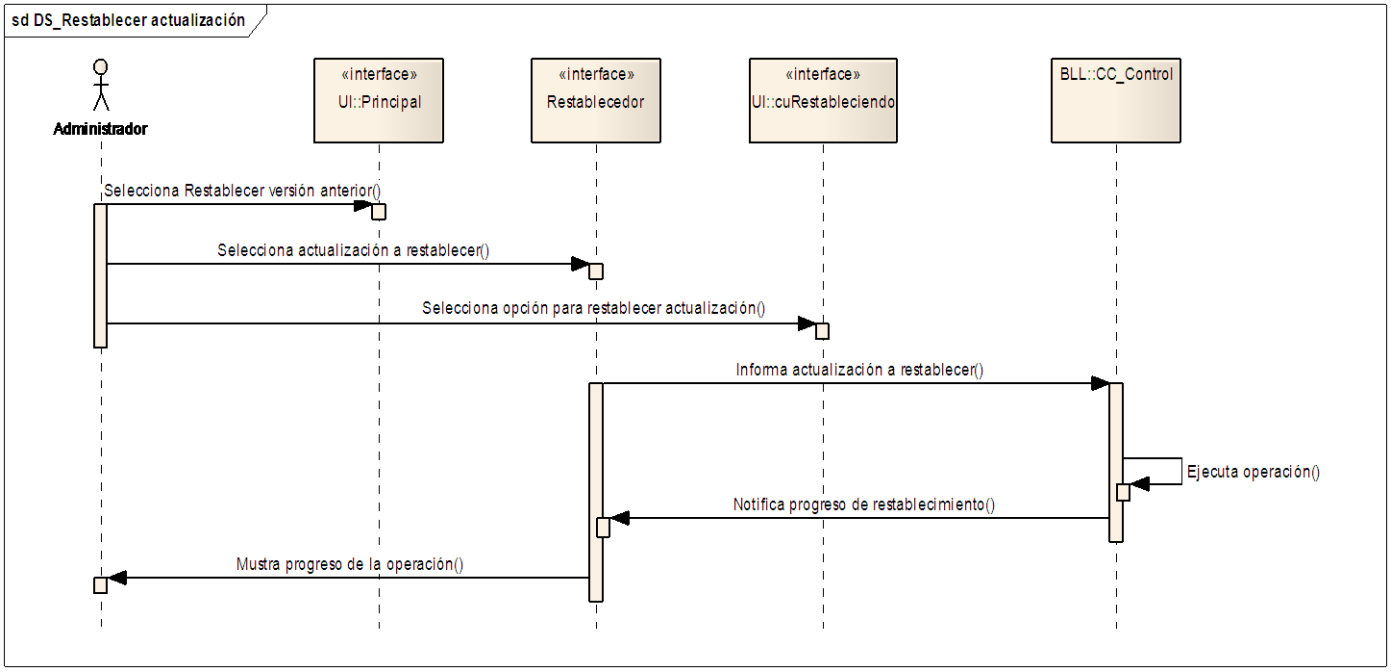

Fig.10 Diagrama de secuencia del CU Restablecer actualización.

3.4 Descripción de las clases principales.

Nombre: controladora "UpdaterManager"
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, realizar copias de seguridad o deshacer actualización.

Nombre: controladora "Control"
Descripción: Es la clase encargada de instanciar y crear un hilo de procesos único para cada actualización, garantizando que se ejecuten cada una de las funcionalidades de la clase controladora Actualizar Aplicación. Su función principal es controlar cada uno de los procesos que levanta una actualización, además de verificar los requerimientos previos y secundarios.

Nombre: interfaz "Principal"
Descripción: Es la interfaz principal del Actualizador Automático. En ella se encuentran las demás interfaces con que cuenta el Actualizador como son: Instalar actualización, Restablecer versión anterior, Buscar actualizaciones, Eliminar archivos temporales y Ver actualizaciones instaladas.

Nombre: interfaz "cuActualizando"
Descripción: Es la interfaz donde se llevan a cabo todas las acciones comprendidas dentro del proceso de actualización. Estas acciones pueden ser: Inicializar la ejecución de las tareas de actualización o restablecimiento, realizar copias de seguridad y deshacer actualización.

Con el desarrollo de este capítulo se obtuvieron los diagramas de clases del análisis, la estructura del diseño mediante el diagrama de paquetes, los diagramas de clases del diseño y los diagramas de secuencia. Cada uno de ellos es de vital importancia, debido a que muestran de una forma más detallada las características que tendrá la aplicación, siendo estos la base para comenzar la implementación del sistema.

Capítulo 4. Implementación.

Con el desarrollo del presente capítulo se pretende mostrar cómo los elementos del Modelo de Diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el Diagrama de despliegue. Durante la implementación se deberá solucionar, a menudo, pequeños problemas relacionados con el entorno de implementación que no deberían afectar al Modelo de diseño.

4.1 Modelo de implementación.

4.1.1 Diagrama de componentes.

Los diagramas de componentes modelan la vista estática de un sistema. Se representan como un grafo de componentes de software unidos por medio de relaciones de dependencia (compilación, ejecución), logrando mostrarse las interfaces que estos soportan. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe una parte del sistema.

En un diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. El uso más importante de un diagrama de componentes es mostrar la estructura del Modelo de implementación, específicamente:

- Subsistemas de implementación y sus dependencias de importación.
- Los subsistemas de implementación organizados en capas.

Componente: Es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación, un conjunto de interfaces y proporciona la realización de los mismos. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos (ficheros ejecutables, binarios, entre otros.). Son las piezas reutilizables de alto nivel a partir de las cuales se pueden construir sistemas. A continuación se muestra el Diagrama de componentes del sistema. Ver (Fig.11).

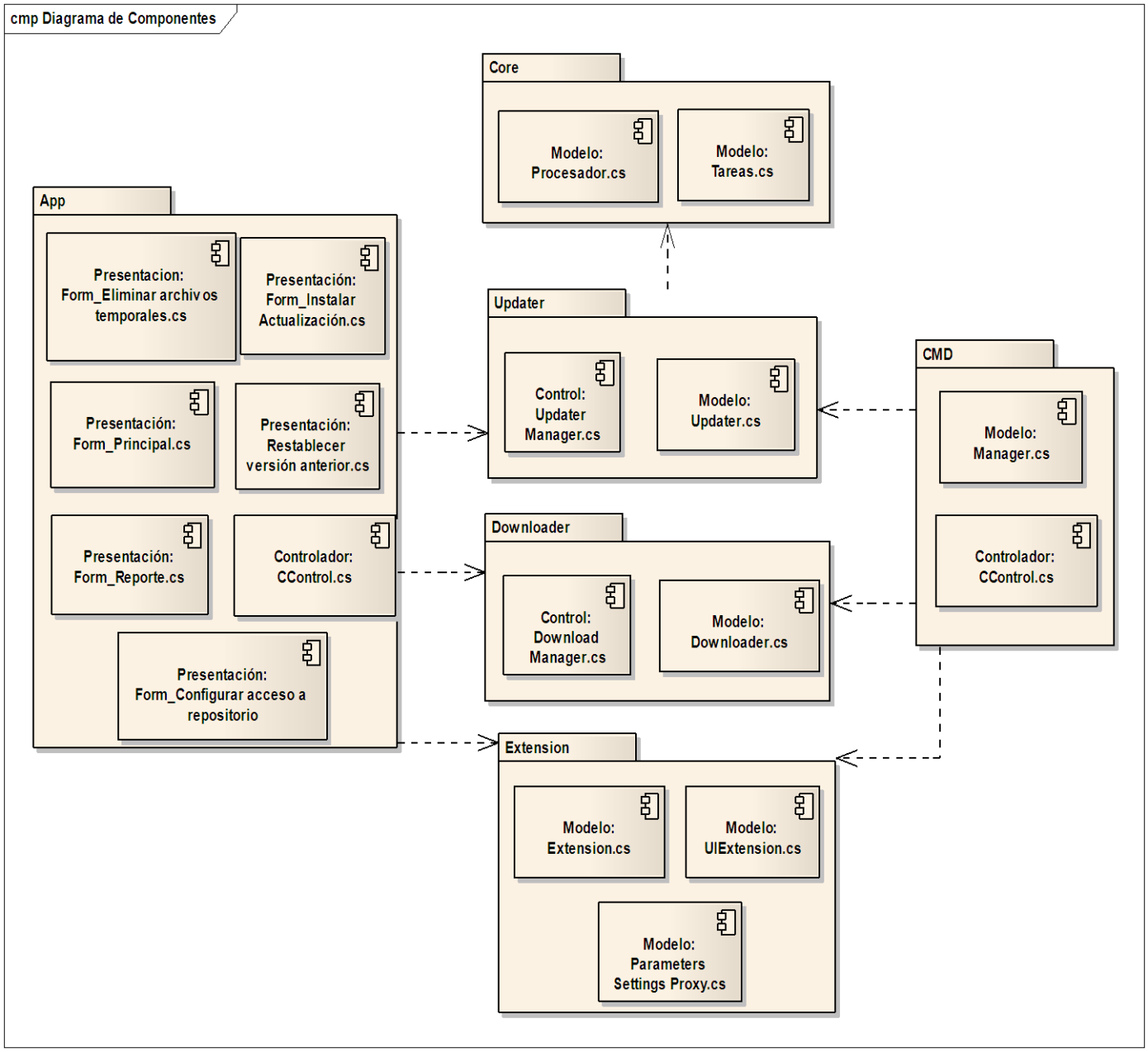

Fig.11 Diagrama de componentes del sistema.

4.1.2 Diagrama de despliegue.

Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representan generalmente un procesador o un dispositivo sobre el que se pueden desplegar los componentes. Ver (Fig.12).

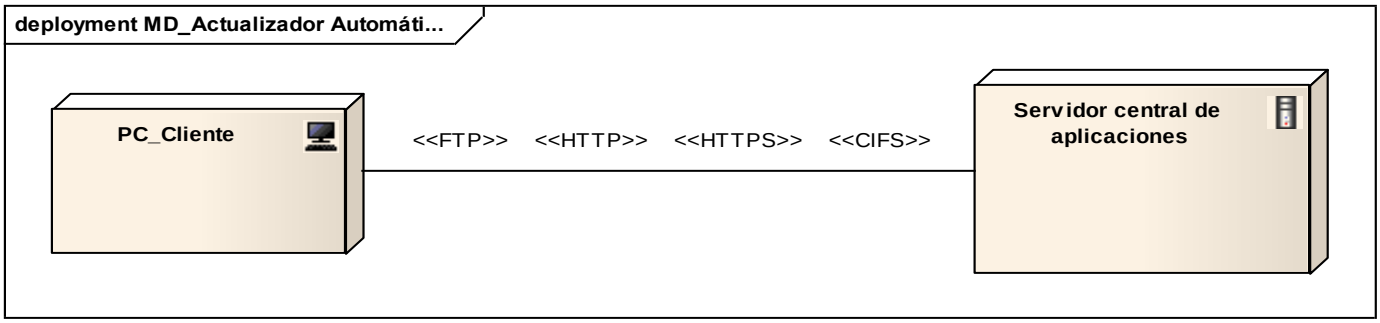

Fig.12 Diagrama de despliegue del sistema.

4.2 Estándares de Codificación.

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código. Un estándar de codificación completo comprende todos los aspectos de la generación de código. Los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso.

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. Es por esto que la codificación del sistema a desarrollar debe cumplir ciertos requisitos, detallados en las presentes tablas.

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 éste 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, éste 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 para el tipo de datos en minúscula.	El nombre que se le da a los controles deben comenzar con las primeras letras en minúscula, las cuales identificaran el tipo de datos al que se refiere, 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

Tabla 4.7 Estándar de codificación. Nomenclatura de los controles visuales.

Al concluir el desarrollo del presente capítulo se obtuvo el Modelo de implementación, el cual se encuentra compuesto por los Diagramas de componentes y de despliegue. De este modo quedan estructurados los elementos de implementación, basándose en las responsabilidades asignadas a los subsistemas de implementación y su contenido. Además se expusieron los estándares de codificación utilizados para el desarrollo del sistema, posibilitando una mejor comprensión del mismo.

CONCLUSIONES

Luego de finalizar la investigación para el desarrollo del presente trabajo se arribaron a las siguientes conclusiones:

- ✓ El análisis de la primera versión del Actualizador Automático, evidenció que éste aún no era un producto estable para la explotación en ambientes de producción real.
- ✓ La aplicación desarrollada cuenta con las principales funcionalidades vinculadas al dominio de los actualizadores automáticos, logrando así un sistema acorde a las tendencias actuales.
- ✓ La integración del intérprete de comandos dota al actualizador de una funcionalidad vital, para su manipulación en servidores UNIX que no cuentan con interfaz gráfica.
- ✓ Con la extensión y migración, el Actualizador Automático brinda sus servicios a una mayor gama de productos que antes no eran soportados.
- ✓ La arquitectura definida inicialmente en la primera versión facilitó la extensión y migración de la aplicación.
- ✓ La solución desarrollada establece la base para la extensión, soporte y mantenimiento de los productos desarrollados en el Centro de Informática Médica.

RECOMENDACIONES

Para lograr la continuidad de este trabajo, debido a la importancia que posee en la actualización de los sistemas desarrollados por el CESIM, se proponen un conjunto de funcionalidades que aumentarán los beneficios proporcionados por el sistema.

- ✓ Implementar un mecanismo que permita al Actualizador Automático optimizar el funcionamiento de los procesos y servicios locales, afectados durante el proceso de actualización.
- ✓ Ampliar la compatibilidad del motor de escaneo del sistema con otros sistemas Antivirus.
- ✓ Dotar al sistema de un módulo que permita la comunicación con procesos remoto, proporcionándole la capacidad de actualizar sistemas remotos.
- ✓ Diseñar la interfaz gráfica del Actualizador Automático utilizando el conjunto de bibliotecas multiplataformas GTK+, para proveer al sistema una interfaz nativa del Framework MONO.

REFERENCIAS BIBLIOGRÁFICAS

1. Ciberaula. [En línea] [Citado el: 25 de octubre de 2010.] http://linux.ciberaula.com/articulo/ventajas_inconvenientes_linux/.

2. definición.de. [En línea] [Citado el: 26 de octubre de 2010.] http://definicion.de/aplicacion/.

3. definición.de. [En línea] [Citado el: 15 de noviembre de 2010.] http://www.definicionde.com/actualizacion-de-software/.

4. definición.de. [En línea] [Citado el: 15 de noviembre de 2010.] http://definicion.de/automatico/.

5. **Delgado Murciano, Anavel y Jarrosay Charadan, Yoelkis.** *Actualizador Automático.* 2010.

6. Diccionario sensagent. [En línea] [Citado el: 16 de noviembre de 2010.] http://diccionario.sensagent.com/int%C3%A9rprete+de+comandos/es-es/.

7. Aula Linux. [En línea] [Citado el: 16 de noviembre de 2010.] http://www.aulalinux.com.ar/shell-o-caparazon-interprete-de-comandos/.

8. **IEEE Computer Society.** *Standard for Software Maintenance.* Los Alamitos.

9. Kioskea. [En línea] [Citado el: 20 de noviembre de 2010.] http://es.kioskea.net/contents/unix/unix-shell.php3.

10. Mailxmail. [En línea] [Citado el: 20 de noviembre de 2010.] http://www.mailxmail.com/curso-linux-unix/linux-unix-funciones-shell.

11. Mailxmail. [En línea] [Citado el: 21 de noviembre de 2010.] http://www.mailxmail.com/curso-linux-unix/linux-unix-nucleo-shell.

12. Iblio. [En línea] [Citado el: 14 de noviembre de 2010.] http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/CURSOLINUX/curso_linux/node66.html.

13. Profesores.elo. [En línea] [Citado el: 16 de noviembre de 2010.] http://profesores.elo.utfsm.cl/Manuel%20Cornejo.pdf.

14. Guía Ubuntu. [En línea] [Citado el: 05 de diciembre de 2010.] http://www.guia-ubuntu.org/index.php?title=Aptitude.

15. Ubuntu.es. [En línea] [Citado el: 25 de noviembre de 2010.] http://doc.ubuntu-es.org/Synaptic.

16. Acerca de Ubuntu. [En línea] [Citado el: 25 de noviembre de 2010.] http://acercadeubuntu.blogspot.com/2009/04/gestor-de-paquetes-synaptic.html.

17. RUP. [En línea] [Citado el: 01 de diciembre de 2010.] http://metodologiaxpvsmetodologiarup.com/.

18. Rational.com. [En línea] [Citado el: 01 de diciembre de 2010.] http://www.rational.com.ar/herramientas/rup.html.

19. Sparxsystems.es. [En línea] [Citado el: 01 de diciembre de 2010.] http://www.sparxsystems.es/New/products/ea_features.html.

20. Microsoft.com. [En línea] [Citado el: 02 de diciembre de 2010.] https://partner.microsoft.com/spain/40013160.

21. Microsoft.com. [En línea] [Citado el: 16 de diciembre de 2010.] http://www.microsoft.com/net/.

22. Fmcancun. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.fmcancun.net/articulos/150-mono-24-y-monodevelop-20-anunciados-oficialmente.html.

23. Mono-Project.com. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.mono-project.com.

24. Ubuntu.es. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.ubuntu-es.org/?q=node/109450.

25. Mentores.net. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.mentores.net/articulos/Mono_Net_Framework_plataformas_noWindows.htm.

26. Sitio Oficial NAnt. [En línea] [Citado el: 02 de diciembre de 2010.] http://nant.sourceforge.net/.

27. Trevinca. [En línea] [Citado el: 04 de diciembre de 2010.] http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node156.html.

28. Desarrolloweb.com. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.desarrolloweb.com/articulos/449.php.

29. UML. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.din.uem.br/.

30. C# Online. [En línea] [Citado el: 05 de diciembre de 2010.] http://es.csharp-online.net/Visi%C3%B3n_general_sobre_CSharp.

31. Scribd.com. [En línea] [Citado el: 05 de diciembre de 2010.] http://www.scribd.com/doc/7411856/Caracteristicas-de-C.

32. **Fernández Cedeño, Karel .** *Documento de Arquitectura de Software.* 2010.

33. Livedocs. [En línea] [Citado el: 25 de marzo de 2011.] http://livedocs.adobe.com.

34. **Visconti, Marcello y Astudillo, Hernán.** *Fundamentos de Ingeniería de Software.* Santa María.

35. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1994.

BIBLIOGRAFÍA

Acerca de Ubuntu. [En línea] [Citado el: 25 de noviembre de 2010.] http://acercadeubuntu.blogspot.com/2009/04/gestor-de-paquetes-synaptic.html.

Aula Linux. [En línea] [Citado el: 16 de noviembre de 2010.] http://www.aulalinux.com.ar/shell-o-caparazon-interprete-de-comandos/.

C# Online. [En línea] [Citado el: 05 de diciembre de 2010.] http://es.csharp-online.net/Visi%C3%B3n_general_sobre_CSharp.

Ciberaula. [En línea] [Citado el: 25 de octubre de 2010.] http://linux.ciberaula.com/articulo/ventajas_inconvenientes_linux/.

definición.de. [En línea] [Citado el: 15 de noviembre de 2010.] http://www.definicionde.com/actualizacion-de-software/.

definición.de. [En línea] [Citado el: 15 de noviembre de 2010.] http://definicion.de/automatico/.

definición.de. [En línea] [Citado el: 26 de octubre de 2010.] http://definicion.de/aplicacion/.

Delgado Murciano, Anavel y Jarrosay Charadan, Yoelkis. *Actualizador Automático.* 2010.

Desarrolloweb.com. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.desarrolloweb.com/articulos/449.php.

Diccionario sensagent. [En línea] [Citado el: 16 de noviembre de 2010.] http://diccionario.sensagent.com/int%C3%A9rprete+de+comandos/es-es/.

Epidata Consulting. [En línea] [Citado el: 27 de noviembre de 2010.] http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=15#Conociendo_UML_2.0.

Estándares de Codificación . [En línea] [Citado el: 07 de 4 de 2011.] http://www.dosmilmastres.com/blog/revisiones-de-cdigo-y-estndares-de-codificacin/.

Estilo y patrón. [En línea] [Citado el: 25 de noviembre de 2010.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

Eva. [En línea] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_10/Conferencia_10/Materiales_complementarios/Introduccion_a_la_Disciplina_Analisis_y_Diseno.pdf.

Fernández Cedeño, Karel . *Documento de Arquitectura de Software.* 2010.

Fmcancun. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.fmcancun.net/articulos/150-mono-24-y-monodevelop-20-anunciados-oficialmente.html.

Gamma, Erich, y otros. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1994.

Garlan, David y Shaw, Mary. *An introduction to software architecture.* New Jersey : V.Ambriola and G.Tortora, World Scientific.

Guía Ubuntu. [En línea] [Citado el: 05 de diciembre de 2010.] http://www.guia-ubuntu.org/index.php?title=Aptitude.

Iblio. [En línea] [Citado el: 14 de noviembre de 2010.] http://www.ibiblio.org/pub/linux/docs/LuCaS/Tutoriales/CURSOLINUX/curso_linux/node66.html.

Ingenieros Software. [En línea] [Citado el: 06 de diciembre de 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

Kioskea. [En línea] [Citado el: 20 de noviembre de 2010.] http://es.kioskea.net/contents/unix/unix-shell.php3.

Livedocs. [En línea] [Citado el: 25 de marzo de 2011.] http://livedocs.adobe.com.

Mailxmail. [En línea] [Citado el: 20 de noviembre de 2010.] http://www.mailxmail.com/curso-linux-unix/linux-unix-funciones-shell.

Mailxmail. [En línea] [Citado el: 21 de noviembre de 2010.] http://www.mailxmail.com/curso-linux-unix/linux-unix-nucleo-shell.

Mentores.net. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.mentores.net/articulos/Mono_Net_Framework_plataformas_noWindows.htm.

Microsoft.com. [En línea] [Citado el: 02 de diciembre de 2010.] https://partner.microsoft.com/spain/40013160.

Microsoft.com. [En línea] [Citado el: 16 de diciembre de 2010.] http://www.microsoft.com/net/.

Mono-Project.com. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.mono-project.com.

Profesores.elo. [En línea] [Citado el: 16 de noviembre de 2010.] http://profesores.elo.utfsm.cl/Manuel%20Cornejo.pdf.

Rational.com. [En línea] [Citado el: 01 de diciembre de 2010.] http://www.rational.com.ar/herramientas/rup.html.

RUP. [En línea] [Citado el: 01 de diciembre de 2010.] http://metodologiaxpvsmetodologiarup.com/.

Scribd.com. [En línea] [Citado el: 05 de diciembre de 2010.] http://www.scribd.com/doc/7411856/Caracteristicas-de-C.

Scribd.com. [En línea] [Citado el: 05 de diciembre de 2010.] http://www.scribd.com/doc/23161581/Estilos-Arquitectonico.

Sitio Oficial NAnt. [En línea] [Citado el: 02 de diciembre de 2010.] http://nant.sourceforge.net/.

Society, IEEE Computer. *Standard for Software Maintenance.* Los Alamitos.

Sparxsystems.es. [En línea] [Citado el: 01 de diciembre de 2010.] http://www.sparxsystems.es/New/products/ea_features.html.

Trevinca. [En línea] [Citado el: 04 de diciembre de 2010.] http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node156.html.

Ubuntu.es. [En línea] [Citado el: 03 de diciembre de 2010.] http://www.ubuntu-es.org/?q=node/109450.

Ubuntu.es. [En línea] [Citado el: 25 de noviembre de 2010.] http://doc.ubuntu-es.org/Synaptic.

UML. [En línea] [Citado el: 04 de diciembre de 2010.] http://www.din.uem.br/.

Visconti, Marcello y Astudillo, Hernán. *Fundamentos de Ingeniería de Software.* Santa María.

GLOSARIO DE TÉRMINOS

- **Aplicación informática:** Es un tipo de software que permite al usuario realizar uno o más tipos de trabajo.
- **Clase:** Conjunto de objetos que comparten atributos, operaciones, relaciones y semántica; representan los conceptos fundamentales del sistema.
- **Diagrama de Clases:** Representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Los diagramas de Clases por definición son estáticos, representan que partes interactúan entre sí, no lo que ocurre cuando.
- **Intérprete de comando:** Es un programa informático que tiene la capacidad de traducir las órdenes que introducen los usuarios, mediante un conjunto de instrucciones facilitadas por él mismo.
- **Lenguaje de Marcado Matemático (MathML, por sus siglas en inglés):** Es un lenguaje de marcado basado en XML, cuyo objetivo es expresar notación matemática de forma que distintas máquinas puedan entenderla, para su uso en combinación con XHTML en páginas web y para intercambio de información entre programas de tipo matemático en general.
- **Lenguaje Extensible de Marcado de Hipertexto (XHTML, por sus siglas en inglés):** Es un lenguaje de programación pensado para sustituir a HTML; XHTML es la versión XML de HTML con las mismas funcionalidades, pero cumple las especificaciones más estrictas de XML.
- **Protocolo de Transferencia de Archivos (FTP, por sus siglas en inglés):** Es un mecanismo que permite la transferencia de archivos a través de una conexión TCP/IP.
- **Sistema Operativo de Disco (DOS, por sus siglas en inglés):** Es el sistema operativo utilizado en la mayoría de los ordenadores personales existentes. Aunque existen diferentes versiones del DOS, la más conocida es la desarrollada por la compañía Microsoft, denominada MS-DOS.
- **Sistema operativo:** Es el software que controla la ejecución de todas las aplicaciones y programas del sistema.
- **SSH:** Es un protocolo seguro y un conjunto de herramientas para reemplazar otras más comunes e inseguras. Fue diseñado desde el principio para ofrecer un máximo de seguridad y permitir el acceso remoto a servidores de forma segura.

- **Terminal:** En informática terminal se refiere al dispositivo hardware usado para introducir o mostrar datos de una computadora. Por extensión puede entenderse como terminal la línea de comandos, que es el software que habitualmente se asocia a estos terminales.
- **UNIX:** Es un sistema operativo multitarea y multiusuario, lo cual significa que puede ejecutar varios programas simultáneamente y que puede gestionar a varios usuarios simultáneamente.

ANEXOS

Descripción de casos de uso del sistema

Caso de Uso: Configurar Aplicación

Caso de Uso:	Configurar Aplicación.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para configurar la aplicación. Éste modifica o crea una nueva configuración, el sistema guarda la configuración y el caso de uso termina.
Precondiciones:	Que el usuario tenga permisos de ejecución.
Referencias	RF1,RF2
Prioridad	**Crítico**

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor introduce el comando para acceder a configurar la aplicación.	
2.	El sistema muestra las opciones de configuración.
3. El actor lleva a cabo la configuración de la aplicación.	
4.	El sistema permite cancelar la configuración. Ver Sección1 “Cancelar Configuración”.
5.	Guarda la configuración.
6.	Termina el caso de uso.

Sección1 “Cancelar Configuración”	
Acción del Actor	**Respuesta del Sistema**
1. El actor cancela la configuración.	

2.	El caso de uso termina.
Prototipo de Interfaz	
Poscondiciones:	La configuración del sistema ha sido realizada de forma exitosa.

Caso de Uso: Actualizar aplicaciones

Caso de Uso:	Actualizar Aplicaciones.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para actualizar las aplicaciones seleccionadas, el sistema lleva a cabo la actualización y el caso de uso termina.
Precondiciones:	Que existan actualizaciones disponibles.
Referencias	RF3
Prioridad	**Crítico**
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor introduce el comando para actualizar una aplicación.	
2.	El sistema muestra las actualizaciones disponibles. Ver caso de uso Mostrar Actualizaciones Disponibles.
3. Introduce parámetros para realizar la actualización.	
4.	El sistema inicia la tarea de descarga del paquete de actualización. Ver caso de uso: Descargar Paquete de Actualización. Comprueba la integridad del paquete de actualización, si es correcta descomprime el paquete, sino Ver

	Sección 1: “Ocurrió un error”. Luego accede al documento “Manifiesto.xml” e inicia el flujo de tareas que comprenden la actualización y realiza una copia de seguridad por cada uno de los ficheros que son modificados. Ver caso de uso: Realizar salva de aplicación. Luego visualiza una lista de los ficheros que están siendo modificados. Completa la actualización y muestra el mensaje “La actualización ha sido exitosa” Permite: “Restablecer” la actualización. Ver caso de uso: Restablecer Actualización.
5.	Termina el caso de uso.
Prototipo de Interfaz	
Sección1 “Ocurrió un error”	
Acción del Actor	**Respuesta del Sistema**
1.	Muestra uno de los siguientes mensajes en pantalla: • “Ha ocurrido un error durante la descarga. Verifique la configuración de la aplicación y los permisos de usuario.” • Error. No se pudo encontrar el archivo o carpeta. • Error. Verifique los permisos de usuario.

	Ejecuta la operación deshacer actualización. Ver caso de uso: Deshacer Actualización.
Prototipo de Interfaz	
Poscondiciones:	La actualización de la aplicación se realizó con éxito.

Caso de Uso: Restablecer actualización

Caso de Uso:	Restablecer Actualización.
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el actor introduce el comando para restablecer una actualización, el sistema comienza a deshacer cada una de las tareas que fueron ejecutadas durante la actualización. Una vez que termine el proceso se le informa al actor y el caso de uso termina.
Precondiciones:	Que existan las copias de seguridad de la aplicación.
Referencias	RF10
Prioridad	**Crítico**
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia cuando el actor introduce el comando para restablecer una actualización.	
2.	Se muestran las aplicaciones disponibles para restablecer. Ver caso de uso: Mostrar Actualizaciones a Restablecer.
3. El actor introduce el identificador de la aplicación o las aplicaciones a restablecer.	
4.	El sistema realiza una salva de la aplicación. Ver caso

	de uso: Realizar Salva de Aplicación. El sistema comienza a deshacer cada una de las tareas que fueron ejecutadas durante la actualización. En caso de ocurrir un error: Ver Sección 1: “Ocurrió un error”.
5.	Termina el caso de uso.
Sección1 “Ocurrió un error”	
Acción del Actor	**Respuesta del Sistema**
1.	Muestra el mensaje: “Ha ocurrido un error durante el restablecimiento. Verifique el estado de la salva”. Ejecuta la operación deshacer actualización. Ver caso de uso: Deshacer Actualización.
2.	Termina el caso de uso.
Prototipo de Interfaz	
Poscondiciones:	Restablecer exitosamente a la versión anterior de una aplicación luego de haber ejecutado su actualización.

2. Diagramas de secuencia.

Diagrama de secuencia del CU Eliminar archivos temporales. Ver (Fig.13)

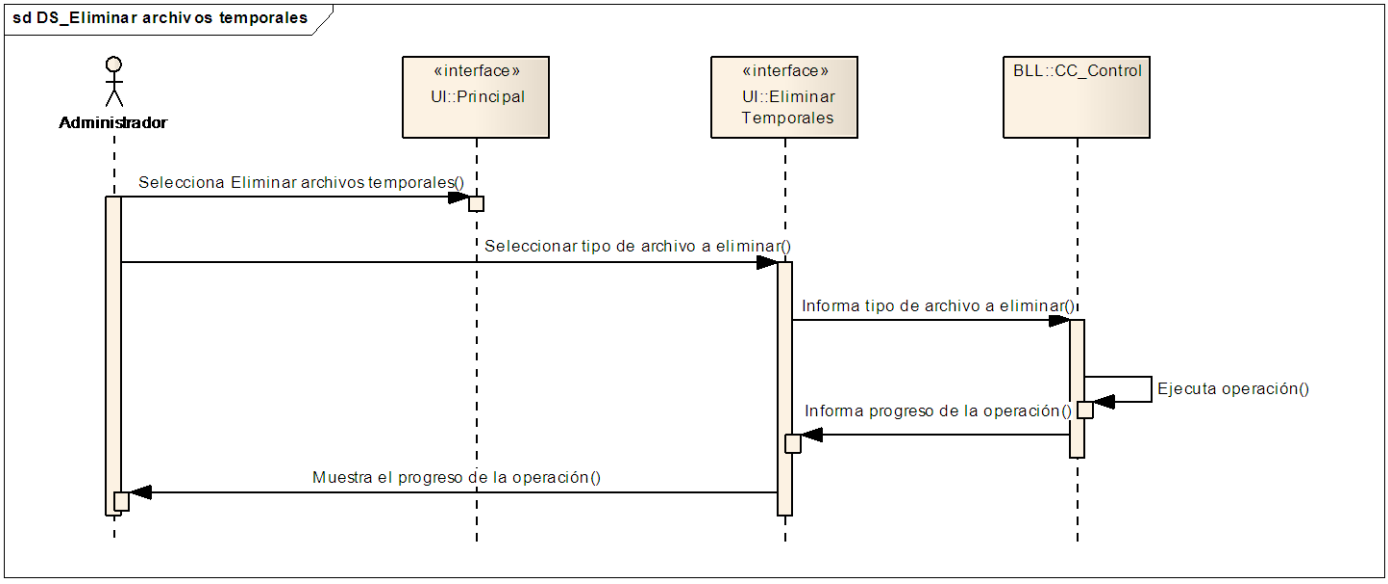

Fig.13 Diagrama de secuencia del CU Restablecer actualización.

Diagrama de secuencia del CU Mostrar información de actualizaciones realizadas. Ver (Fig.14)

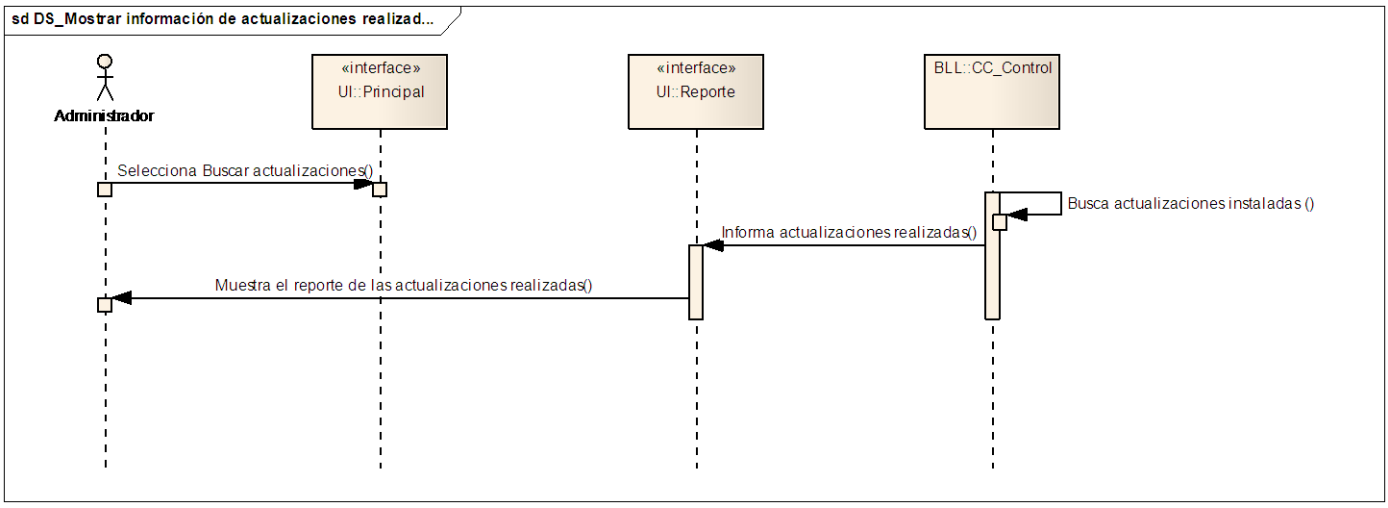

Fig.14 Diagrama de secuencia del CU Mostrar información de actualizaciones realizadas.