Universidad de las Ciencias Informáticas
Facultad 3

Propuesta de un sistema de actualizaciones automáticas para los Registros y Notarías de la República Bolivariana de Venezuela.

Trabajo de Diploma para optar por el título de

Ingeniero Informático.

Autores: Angel Alberto Bello Caballero.

Manuel Alejandro Diaz Alonso.

Tutor: Ing. Armando Esteban Pacheco Iglesias.

La Habana, Cuba.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al centro CEGEL de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____ días del mes de _____ del año 20___.

Angel Alberto Bello Caballero

_____

Firma del Autor

Manuel Alejandro Diaz Alonso

_____

Firma del Autor

Ing. Armando Esteban Pacheco Iglesias

_____

Firma del Tutor

DATOS DE CONTACTO

Ing. Armando Esteban Pacheco Iglesias

aeiglesias@uci.cu

AGRADECIMIENTOS.

A nuestros compañeros de trabajo y de convivencia en esta universidad por soportarnos y ayudarnos en todo momento.

A nuestro tutor por el apoyo brindado durante la realización de esta investigación.

A todos los que de una forma u otra han contribuido a nuestra formación profesional durante estos años de estudio.

A nuestros amigos de Las Tunas y Santiago de Cuba.

DEDICATORIA.

A mi familia por su apoyo incondicional durante estos 5 años de largos estudios y separación, en especial a mi hermano Luis E. Bello Caballero por ser mi ejemplo a seguir y a María C. Mateo por soportarme, apoyarme y estar conmigo siempre.

A Feyse Enrique Caballero Hernández, hombre, padre y abuelo ejemplar. Gracias tu educación y ejemplo y por apoyarme y guiarme hasta el último momento. Que descanses en paz.

Angel Alberto Bello Caballero.

A mi padre y mi madre, por todo lo que se han sacrificado por sus hijos durante toda su vida, sabiendo balancear entre familia y profesión. Por ser excelentes padres y aún mejores personas.

A mi hermano, por cargar el peso del mundo sobre sus hombros desde tan temprano. Por ser alguien que conoce el significado de sacrificio y responsabilidad.

A mi familia, por estar siempre ahí aun cuando yo no he estado.

Manuel Alejandro Diaz Alonso.

RESUMEN

El presente trabajo propone una estrategia de cómo eliminar los problemas actuales existentes a la hora de ejecutar las actualizaciones de los subsistemas automatizados del Servicio Autónomo de Registros y Notarias de la República Bolivariana de Venezuela y las oficinas adscriptas a esta entidad a partir del análisis, diseño e implementación de un sistema que satisfaga las necesidades actuales.

La actualización de los subsistemas de SAREN son de suma importancia pues de esto depende el buen funcionamiento de las gestiones que se realicen en cada una de las oficinas de la entidad, el mantenimiento de la seguridad de los subsistemas, así como mejoras en los software o variaciones solicitadas por SAREN en cuanto a la forma en que se implementan el flujo de trabajo de cada subsistema. Al mantener el sistema actualizado en las oficinas desplegadas se garantiza que todas trabajen con la misma versión del software y se mantengan perfectamente sincronizadas con el Centro de Datos, con el cual intercambian información constantemente.

Se hizo un estudio de herramientas para desarrollar los artefactos necesarios con el objetivo de obtener un producto que satisfaga las necesidades actuales y erradique las deficiencias en el software que actualmente se encuentra llevando a cabo las actualizaciones en las oficinas de SAREN.

PALABRAS CLAVE

Sistema de Actualizaciones Automáticas, Daemon.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La informatización de los procesos de gestión del Servicio Autónomo de Registros y Notarías (SAREN) surge en el año 2005 y está siendo desarrollado en la Universidad de las Ciencias Informáticas. Este responde a las necesidades de la Ministerio del Poder Popular para Relaciones Interiores y Justicia de la República Bolivariana de Venezuela. SAREN cuenta con 7 subsistemas informáticos desplegados o desplegándose en la República Bolivariana de Venezuela, los cuales se listan a continuación: Portal SAREN, Registros Públicos, Registros Mercantiles, Servicio Autónomo Gerencial, Servicio Autónomo de Gestión, Administración Financiera y Digitalización y Metadatos.

La ejecución y puesta a punto del proyecto de colaboración para los Registros y Notarías de la República Bolivariana de Venezuela ha contribuido de manera significativa al seguimiento de la política de gobierno encaminada al mejoramiento de la calidad de vida de la sociedad venezolana, lo cual se considera un aporte importante al desarrollo social de esta nación. Teniendo en cuenta la situación previa existente, el impacto alcanzado por la aplicación de la solución informática de este proyecto de gestión integral ha sido:

En la primera fase hasta la fecha existen más de 200 oficinas registrales desplegadas. De este modo se comienza a revertir el hecho de que el 90% de las oficinas de registros y notarías no poseían una infraestructura tecnológica apropiada, por lo que la información que recibía la Dirección Nacional de Registros y Notarías no era oportuna y confiable, lo cual afectaba en gran medida el buen funcionamiento, organización y administración del Sistema de Registros y Notarías.

La solución de software ha permitido, en gran medida, estandarizar los procesos asociados al funcionamiento de las oficinas de los Registros Públicos y Mercantiles de Venezuela y los procesos financieros y contables de todas las oficinas con la implantación del sistema de Administración Financiera, lo cual afectaba en gran medida el principio de legalidad establecido por las leyes venezolanas.

Se equilibraron las condiciones de trabajo en todas las oficinas registrales desplegadas, fundamentalmente en los aspectos tecnológicos y de seguridad física, lo cual afectaba de forma directa el funcionamiento y la calidad del servicio prestado al ciudadano.

Se crea un sistema centralizado que organiza toda la información a nivel nacional y apoya a la toma de decisiones del gobierno venezolano. Esto se dificultaba con el proceso manual previo existente provocando que el acceso a la información de los registros fuera muy engorroso, difícil y de carácter regional, por lo que la publicidad registral se afectaba.

Periódicamente los subsistemas de SAREN son actualizados con versiones realizadas por el equipo de desarrollo, por lo que es necesario tener una forma de aplicar estas actualizaciones a cada una de las oficinas desplegadas. El mecanismo más viable para llevar a cabo esta tarea es mediante un sistema de actualizaciones automáticas.

Hoy en día la actualización del sistema SAREN se lleva a cabo mediante un software de actualización realizado por el equipo de trabajo del proyecto Servicio Autónomo de Identificación, Migración y Extranjería (SAIME) en el año 2006 en Microsoft .NET Framework 1.1 para Windows XP, el cual ha sido poco actualizado desde su liberación. Este software es ineficiente en cuanto a conexión, transmisión de datos y en reiteradas ocasiones queda disfuncional debido a la ocurrencia de errores en tiempo de ejecución.

En aras de eliminar estos inconvenientes se emprendió como tarea fundamental la creación de un producto informático que permita las actualizaciones automáticas de los diferentes subsistemas de SAREN, el cual sea un mecanismo seguro para realizar despliegues centralizados de software.

La tendencia actual a nivel mundial de desarrollo va dirigida a implementar software que incorpore actualizaciones automáticas. Estos sistemas constan de dos aplicaciones, servidor y cliente. Con ello se evita el gasto innecesario de tiempo y recursos al evitar que exista un equipo de soporte que se dedique a mantener actualizado el software en cada una de las computadoras instaladas o que el usuario final sea quien tenga que realizar dicha operación, impidiendo así posibles errores debido a su falta de experiencia. De ahí que este producto sea utilizado por todo tipo de software, desde programas antivirus hasta gestores de paquetes y sistemas operativos.

En Cuba el tema de actualizaciones automáticas no ha alcanzado un desarrollado elevado, sin embargo existen algunos sistemas que utilizan este mecanismo para mantenerse actualizados, un ejemplo de ello es el antivirus Sav32. Debido a la infraestructura de las redes en Cuba la tendencia no va

dirigida a la implementación de actualizaciones automáticas, sino que se realizan parches para ser distribuidos e instalados por un equipo de soporte o por el usuario final.

En la República Bolivariana de Venezuela existe una gran inestabilidad en la red de la cual hace uso SAREN. Cuando se está llevando a cabo el proceso de actualización de los subsistemas registrales y ocurre una falla en la red, el actual software no es capaz de recuperarse ni de llevar el registro de cuál fue el último archivo descargado y debido al mal tratamiento de excepciones que tiene implementado la aplicación queda sin respuesta e inactiva.

Esta situación trae como consecuencia que se tenga que reiniciar la aplicación manualmente varias veces, con la consecuente pérdida de tiempo y dinero, ya que debe existir un personal encargado de configurar cada una de los servidores y clientes de las oficinas que fueron afectadas por el fallo.

Problema a Resolver:

¿Cómo lograr una mayor eficiencia en los procesos de actualizaciones automáticas de los subsistemas registrales en el manejo de los errores de conexión, contribuyendo a un menor costo para los Registros y Notarías de la República Bolivariana de Venezuela?

Objeto de estudio:

El proceso de desarrollo de software.

Campo de acción:

Proceso de actualizaciones automáticas de los subsistemas de los Registros y Notarías de la República Bolivariana de Venezuela.

Objetivo General:

Desarrollar un servicio de actualizaciones automáticas para lograr una mayor eficiencia en el control de las conexiones y descargas de archivos, contribuyendo así a un mejor aprovechamiento de tiempo y recursos de las entidades adscriptas a la dirección de los Registros y Notarías de la República Bolivariana de Venezuela.

Objetivos Específicos:

1. Analizar los estilos arquitectónicos, herramientas y tecnologías existentes para concretar el estudio del estado del arte de los sistemas de actualizaciones automáticas.
2. Modelar los procesos de negocio que comprenden un sistema de actualizaciones automáticas para comprender las funcionalidades necesarias para su correcto funcionamiento.
3. Realizar el diseño y la implementación de los procesos de negocio del sistema de actualizaciones automáticas que permitan que el sistema sea capaz de recuperarse ante fallas y llevar el registro de los eventos y errores que ocurran durante su ejecución.
4. Validar el sistema de actualizaciones automáticas propuesto en cuanto a eficiencia y calidad en la transmisión de datos y su recuperación ante fallas.

Idea a Defender:

El desarrollo de un sistema de actualizaciones automáticas para los subsistemas registrales logrará una mayor eficiencia en el control de las conexiones y descargas de archivos contribuyendo a un mejor aprovechamiento de tiempo y recursos de las entidades adscriptas a la dirección de los Registros y Notarías de la República Bolivariana de Venezuela.

Tareas a resolver:

1. Realización de un estudio del estado del arte de los procesos que se llevan a cabo para las actualizaciones automáticas de datos.
2. Realización de un estudio de las diferentes tecnologías y procedimientos que se llevan a cabo en los servicios de actualización automáticas.
3. Realización de un estudio de compatibilidad de sistemas operativos para los servicios de actualización automática.
4. Realización de la captura de requisitos de los procesos que se deseen automatizar.
5. Realización de una solución de diseño que permita que el sistema sea confiable y seguro para el manejo de los datos.
6. Implementación de una solución informática que permita que el servicio de actualizaciones automática que lleve un control de cada evento.

7. Verificación de la robustez de los diseños y del software en general, mediante métricas a los diagramas de diseños y pruebas de Caja Negra al software.

Posibles Resultados:

Solución de un producto informático que permita las actualizaciones automáticas de los diferentes subsistemas del SAREN, el cual sea un mecanismo seguro en cuanto al manejo y control del proceso de actualización.

La investigación está compuesta por cuatro (4) capítulos fundamentales en los cuales se profundizará posteriormente, estos tratan desde la investigación hasta la validación de los temas tratados y propuestos por el equipo de desarrollo.

El *Capítulo 1* aborda la fundamentación teórica para la realización de la solución propuesta, para la cual se tienen en cuenta el estado del arte de las actuales técnicas de programación y metodologías existentes para el desarrollo de software, al igual que un estudio de las plataformas que se pueden utilizar para el desarrollo del sistema.

El *Capítulo 2* aborda la propuesta del sistema, para la cual se tienen en cuenta un estudio de los distintos estilos arquitectónicos y patrones de diseño, fundamentando los escogidos por el equipo de desarrollo para la solución del sistema. Se realiza un análisis de los procesos más importantes dentro del sistema.

El *Capítulo 3* aborda el diseño e implementación de la propuesta que se brinda en el capítulo anterior, esta sección está compuesta por varias definiciones prácticas de los modelos de diseño y la realización de algunos de los diagramas de clases y componentes más importantes dentro de los procesos que se proponen en el capítulo anterior.

El *Capítulo 4* aborda la validación del sistema propuesto. En este capítulo se trata todo lo relacionado con las pruebas realizadas al software para la verificación de su calidad.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En este capítulo se tratarán conceptos necesarios para un mejor entendimiento del funcionamiento de los sistemas de actualizaciones automáticas. También se abordará una panorámica de sistemas similares a nivel mundial y en Cuba y las posibles herramientas y metodologías de desarrollo a utilizar.

1.1 Sistemas de Actualizaciones Automáticas.

Para comprender el tema de actualizaciones automáticas, se necesita conocer que es actualización; según la Real Academia Española:

- Hacer actual algo, darle actualidad.
- Poner al día.

La Universidad de las Américas Puebla define actualizar desde el punto de vista informático como el proceso de poner al día un programa incorporándole nuevas características que lo mejoran y lo hagan más estable. (Universidad de las Américas Puebla, 2010)

Según Canonical[1] actualizar, no es más que el proceso de pasar de una versión anterior de un software determinado a una nueva. (Doc.ubuntu-es, 2010)

Microsoft mantiene su sistema operativo Windows actualizado mediante las actualizaciones automáticas, Windows comprueba de forma habitual si hay actualizaciones que puedan contribuir a su mejor desempeño. Cuando son activadas las actualizaciones automáticas, el usuario no tiene que buscar las actualizaciones en línea ni preocuparse de que pueda perderse alguna corrección crítica. Windows las descarga e instala automáticamente. (Microsoft Technet, 2010)

Tomándose en cuenta los conceptos previos, un sistema de actualizaciones automáticas debe estar compuesto de al menos dos software, un servidor y un cliente para realizar la transmisión de los datos que se desean actualizar de forma automática, sin necesidad de que exista un personal capacitado atendiendo este proceso.

[1] Empresa encargada del desarrollo del sistema operativo Ubuntu, el cual es una distribución muy utilizada basada en GNU/Linux.

Según (Universidad de las Américas Puebla, 2010), las actualizaciones, de acuerdo a su nivel de importancia, se dividen en:

Actualizaciones críticas e importantes: Son mejoras para corregir fallos en la seguridad del sistema o errores en los programas y todas deben ser instaladas.

Actualizaciones recomendadas: Son aquellas que se pueden no instalar, siempre y cuando se esté seguro que no se van a utilizar. Son actualizaciones que suponen una mejora sobre el sistema, aunque no influye directamente en su desempeño o seguridad.

Actualizaciones opcionales: Se dividen en actualizaciones de software y actualizaciones de hardware, comprenden desde las actualizaciones de controladores hasta los paquetes opcionales de idiomas.

1.2 Configuración de la red de SAREN.

La red de la cual hace uso SAREN es una red privada virtual, por sus siglas en ingles VPN[2]. Esto permite que todo el tráfico que se genere dentro de esta red sea cifrado, garantizando la seguridad de los datos. La configuración adoptada por las estaciones de trabajo y servidores para conectarse entre sí es una topología de estrella: esta se une en un único punto, normalmente con control centralizado, como un concentrador de cable, todas las estaciones se conectan al concentrador y las señales son distribuidas a todas las estaciones específicas del concentrador. Además presenta una topología lineal: en esta todas las estaciones se conectan directamente a un único canal físico (cable) de comunicación (bus). Es posible unir varios segmentos de buses en una configuración de múltiples bus siendo necesario utilizar repetidores de señal en el caso de grandes distancias. La red del SAREN presenta un híbrido de ambas topologías.

[2] Virtual Private Network

1.3 Necesidad de implementación de un sistema de actualizaciones automáticas para la eficiencia de los procesos de las entidades de SAREN.

Debido a la mala conectividad e inestabilidad de la red de la cual hace uso SAREN, existen ineficiencias en cuanto a transmisión de datos y en reiteradas ocasiones ocurren fallas en la conexión. Si estas fallas ocurren durante la ejecución del proceso de actualización, ocurren errores en el software y en reiteradas ocasiones queda disfuncional. En estos casos debe trasladarse hasta el terminal cliente un personal capacitado que restaure la aplicación a su estado original. La aplicación no lleva un registro de descargas, por lo que cuando se produce el fallo, debe reiniciarse desde el primer archivo. Todo esto conlleva a gastos en tiempo y recursos a la entidad.

1.4 Sistemas similares.

Actualmente las actualizaciones automáticas son utilizadas por diversos sistemas para mantener su software actualizado. Grandes compañías emplean este método para mantener actualizados sus sistemas. Microsoft lo emplea para sus sistemas operativos mediante una herramienta que trae por defecto llamada Windows Update. Adobe para la versión del producto Acrobat Reader 9.1 introdujo este método para mantener actualizado su software. Canonical incluye este método como parte de su sistema para mantener actualizado los paquetes instalados.

Los sistemas antivirus también hacen uso de las actualizaciones automáticas para mantener sus bases de datos actualizados, lo cual es muy importante en este tipo de programas, debido a la importancia que tienen para la protección de los datos y del sistema operativo. Uno de estos sistemas antivirus que hace uso de las actualizaciones automáticas es el Sav32 de fabricación cubana. Para efectuar la actualización de la base de datos, este sistema se conecta a un servidor FTP[3] para así obtener la actualización de la base de datos.

En la Universidad de la Habana se concluyó el diseño del proyecto SADIES[4]. Entre los objetivos principales del proyecto se encuentran: agilizar todo el proceso de asignación de carreras y publicación más temprana de sus resultados y permitir el acceso inmediato a los datos para la realización de estudios sociales sobre la composición del ingreso. (Centro de Estudios para el Perfeccionamiento, 2009)

[3] File Transfer Protocol.
[4] Sistema Automatizado Distribuido de Ingreso a la Educación Superior.

En la Universidad de las Ciencias Informáticas existen programas que hacen uso de este método, como por ejemplo sistema operativo Nova basado en GNU/Linux. No solo se han hecho uso de estos sistemas sino que se han implementado algunos como el Sistema de Actualización del proyecto SAIME, desarrollado por ese equipo de trabajo para la actualización de su software y utilizado actualmente por el sistema de SAREN.

1.5 Metodologías de desarrollo.

Para el desarrollo de un proyecto de software se puede realizar siguiendo una metodología de desarrollo, lo cual es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a producir nuevo software. (Velthuis, 1996)

Las metodologías ágiles se basan en que la capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan. Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados. Es más importante crear un producto que funcione a escribir documentación exhaustiva. La capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan. La colaboración con el cliente debe prevalecer sobre la negociación de contratos. (Figueroa, 2008)

La siguiente tabla expone una comparación de las metodologías de desarrollo (Figueroa, 2008):

Diferencias entre Metodología Tradicionales y Ágiles.

Metodologías Tradicionales	Metodologías Ágiles
Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.	Basadas en heurísticas provenientes de prácticas de producción de código.
Cierta resistencia a los cambios.	Especialmente preparados para cambios durante el proyecto.
Impuestas externamente.	Impuestas internamente (por el equipo).
Proceso mucho más controlado, con numerosas políticas/normas.	Proceso menos controlado, con pocos principios.

El cliente interactúa con el equipo de desarrollo mediante reuniones.	El cliente es parte del equipo de desarrollo.
Más artefactos.	Pocos artefactos.
Más roles.	Pocos roles.
Grupos grandes y posiblemente distribuidos.	Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.
La arquitectura del software es esencial y se expresa mediante modelos.	Menos énfasis en la arquitectura del software.
Existe un contrato prefijado.	No existe contrato tradicional o al menos es bastante flexible.

Tabla 1. Comparación entre las metodologías tradicionales y ágiles.

En la metodología ágil **Scrum** se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, los requisitos son cambiantes o poco definidos. La innovación, la competitividad, la flexibilidad y la productividad son fundamentales (eumet.net, 2009).

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto. (eumet.net, 2009)

Otra metodología ágil muy utilizada es **XP**[5], la cual es una metodología basada en prueba y error. Los requisitos pueden (y van a) cambiar. Necesita de un grupo de desarrollo pequeño y muy integrado y con formación elevada y capacidad de aprender. Debe tener un cliente bien definido. Esta se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. XP surgió como respuesta y posible solución a los problemas derivados del cambio en los requerimientos, se plantea como una metodología a emplear en proyectos de riesgo. (Escribano, 2002)

[5] eXtreme Programming (Programación Extrema)

MSF[6] es una metodología flexible e interrelacionado grupo de conceptos, modelos y mejoras prácticas de uso que controlan la planeación, el desarrollo y la gestión de proyectos tecnológicos. Se centra en los modelos de procesos y de equipo. Permite desarrollar escalabilidad de proyecto pequeños a proyectos extensos y complejos. Consta de 5 fases: Visión y Alcance, Planificación, Desarrollo, Estabilización e Implantación. (Becerra Guzman, 2009)

La metodología **RUP**[7] posee una extensa documentación de referencia en la red y en soporte impreso. Hace énfasis en la documentación como elemento primordial que todo proyecto debe generar al contrario de otras de enfoque más ágil, esto propicia que en próximos ciclos de vida el equipo de desarrollo pueda retroalimentarse.

Un aporte fundamental de esta metodología al desarrollo del trabajo es su enfoque en los casos de uso como guía de todo el proceso de desarrollo. Los casos de uso permiten definir las funcionalidades del sistema en base a las necesidades del cliente, además de tener una trazabilidad de todos los artefactos generados en el ciclo de vida del proyecto. (Pressman, 2008)

1.6 Herramientas CASE.

Existen múltiples herramientas CASE[8] creadas para el desarrollo de la ingeniería de software. Estas existen con el fin de desarrollar programas utilizando técnicas de diseño y metodologías bien definidas, soportadas por herramientas automatizadas. Algunas de estas herramientas son: ArgoUML, Poseidon, MagicDraw UML, BorlandTogether, Microsoft Visio y Visual Paradigm, de esta última la Universidad de las Ciencias Informáticas posee una licencia para su uso.

Visual Paradigm para UML[9] es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Visual Paradigm también proporciona abundantes tutoriales

[6] Microsoft Solution Framework.
[7] Rational Unified Process (Proceso Unificado de Rational).
[8] Computer Aided Software Engineering
[9] Unified Modeling Language

de UML, demostraciones interactivas de UML y proyectos UML, soporta UML versión 2.1, modelado colaborativo con CVS[10] y Subversion. Contiene interoperabilidad con modelos UML2 a través de XMI[11] y posee generación de código. Contiene además editor de detalles de casos de uso, transformación de diagramas de Entidad-Relación en tablas de base de datos, reorganización de las figuras y conectores de los diagramas UML, importación y exportación de ficheros XMI e integración con Microsoft Visio. (Visual Paradigm, 2000)

1.7 Plataformas de desarrollo

Para la implementación de un sistema informático que de solución a la problemática antes planteada es necesario seleccionar una plataforma de desarrollo que reúna varias características acordes a los intereses del sistema y de las personas e instituciones que se beneficiarán del mismo.

Una plataforma de desarrollo es el entorno común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo, sin embargo, también es posible encontrarlas ligadas a una familia de lenguajes de programación o a una Interfaz de Programación de Aplicaciones o API[12]. Se le denomina plataformas a los diferentes ambientes creados para el desarrollo de software. (Ribao, 2006)

Actualmente en el mundo existe una gran variedad de plataformas de desarrollo para aplicaciones web y de escritorio. En los últimos años las plataformas de desarrollo han ido evolucionando siempre con el objetivo de ofrecer una interfaz más amigable para el programador y abstraerlo de las funciones más elementales.

1.6.1 Entorno de desarrollo integrado.

En los últimos años se han desarrollado múltiples IDE[13] que soportan una gran cantidad de lenguajes de programación, sobresaliendo en la actualidad el Visual Studio de Microsoft, destacándose por su framework que unifica varias plataformas como son la .NET y Java, Eclipse de la Eclipse

[10] Concurrent Versions System
[11] XML Metadata Interchange
[12] Application Programming Interface.
[13] Integrated Development Environment.

Foundation sobresaliendo por los módulos (en inglés plug-in) que le brindan extensas funcionalidades y NetBeans que es un IDE de código abierto, totalmente libre y posee una amplia comunidad de desarrollo.

NetBeans es compatible con los sistemas operativos Windows, GNU/Linux, Mac OS X y Solaris. Incluye la creación de proyectos, asistencia de código y depuración. Además incluye la edición y depuración de código PHP y acceso a bases de datos. Posee soporte para desarrollo web, incluyendo JavaScript, Ajax, Ruby, JRuby, Rails, Groovy y Python. NetBeans incluye plantillas para el desarrollo en los lenguajes C/C++. Permite el trabajo sobre estos lenguajes de programación con librerías estáticas y dinámicas. El editor de C/C++ en NetBeans tiene un completamiento de código avanzado y permite definir estilos de código, además está perfectamente integrado a Qt 4 Designer, la cual es una herramienta para el diseño de interfaces gráficas. (NetBeans, 2001)

1.6.2 Lenguaje de programación

El lenguaje de programación C, fue desarrollado por Kernighan y Ritchie en 1972, en la compañía AT&T[14]. C++ surge en 1980. Su autor B. Stroustrup trabajador también de la AT&T, lo comenzó a desarrollar como una extensión del lenguaje C y fue llamado en un principio *C with clases.* El nombre actual del lenguaje C++ hace referencia al carácter del operador de incremento ++.

El lenguaje C++ es un lenguaje versátil, potente y genérico. Mantiene las ventajas del lenguaje C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. También elimina algunas limitaciones del C. El C++ es a la vez un lenguaje procedimental y orientado a objetos, esta característica le brinda compatibilidad con C. C++ soporta y optimiza el trabajo con todo tipo de ficheros incluidos los XML[15]. (Schildt, 1995)

El XML proviene de un lenguaje creado por IBM en los años 70. El lenguaje de IBM se llama GML[16] y surgió por la necesidad que tenían en la empresa de almacenar grandes cantidades de información de temas diversos. (DesarrolloWeb.com, 2001)

[14] American Telephone and Telegraph. Compañía americana de servicios telefónicos.
[15] Extensible Markup Language (Lenguaje de Marcas Extensible).
[16] General Markup Language.

La W3C[17] comenzó a desarrollar el XML en el año 1998 y continúa mejorándolo en la actualidad. Este lenguaje pretende eliminar las carencias del lenguaje HTML en lo que respecta al tratamiento de la información, dándole solución a deficiencias como: la mescla de contenido con los estilos que se quieren aplicar a la información, HTML no permite compartir información con todos los dispositivos existentes como ordenadores o teléfonos móviles y la presentación en pantalla de la información que depende del visor que se utilice.

XML es una tecnología sencilla, que tiene a su alrededor otras tecnologías que la complementan y la hacen mucho más grande y brinda mayores posibilidades a la hora de trabajar con este lenguaje. La principal novedad de XML consiste en permitir compartir los datos con los que se trabaja a todos los niveles, por todas las aplicaciones y soportes. Por esto juega un papel importantísimo en el mundo actual, que tiende a la globalización y la compatibilidad entre los sistemas.

El lenguaje XML permite manipular datos requiriendo integridad, autenticación y privacidad de los mismos. Esto es posible mediante firmas digitales, procesando reglas y sintaxis. Además permite la encriptación de datos y representa estos en formato XML, pueden ser encriptados datos que tengan un documento XML o cualquier formato en específico que se necesite proteger. El resultado de la encriptación es un elemento XML que contiene o hace referencia a los datos cifrados. Mediante el estándar XKMS (XML Key Management Specification) permite especificar protocolos para la distribución y registro de llaves públicas. El XKMS se complementa con las firmas XML y el estándar de encriptación XML (W3C, 2011).

1.6.3 Compilador

La sigla GCC significa *GNU Compiler Collection*. Originalmente significaba *GNU C Compiler*. Este compilador se encuentra en su versión 4.5.1 liberada el 31 de julio de 2010. Es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr. (The GNU Compiler Collection, 2010)

[17] World Wide Web Consortium, Consorcio encargado de la estandarización de internet.

El compilador GCC permite crear ejecutables con enlaces estáticos: los binarios de las funciones se incorporan al código binario del ejecutable o dinámicos: el código de las funciones permanece en la biblioteca. El ejecutable cargará en memoria la biblioteca y ejecutará la parte de código correspondiente en el momento de correr el programa. El enlazado dinámico permite crear un ejecutable más pequeño en cuanto a tamaño del fichero, pero requiere disponible el acceso a las bibliotecas en el momento de ejecutar el programa. El enlazado estático crea un programa autónomo, agrandando el tamaño del ejecutable binario. (The GNU Compiler Collection, 2010)

1.8 Conclusiones

En este capítulo se hizo un estudio de los aspectos que se desean implementar para resolver los problemas existentes en el proceso de actualización de los subsistemas de SAREN, tanto la aplicación para la gestión de las actualizaciones del centro de datos como el software encargado de actualizar los subsistemas en los servidores de las oficinas.

Se llevo a cabo un estudio del funcionamiento de los procesos de los sistemas de actualizaciones automáticas y algunos de los principales sistemas que actualmente hacen uso de este mecanismo, definiéndose la creación de dos software para llevar a cabo este proceso.

Se realizó un análisis de las plataformas existentes para el desarrollo de software y después de haberse analizado todas las ventajas y desventajas que estas ofrecen se decidió escoger como IDE NetBeans porque es robusto y potente en cuanto a soporte de diferentes tecnologías, es un IDE multiplataforma y tiene un buen completamiento de código lo cual agiliza el desarrollo.

El compilador GCC fue el escogido ya que posee una correcta integración con NetBeans y permite la depuración de código. Este compilador posee una amplia colección de librerías para C/C++.

También se hizo un análisis de las metodologías y las herramientas y se decidió escoger RUP, complementado por la herramienta CASE Visual Paradigm para UML. Se realizó esta elección por las características que posee el software y por todos los contratos existentes que exige el cliente, o sea, se debe entregar una documentación bastante extensa la cual es facilitada por la metodología escogida. La herramienta CASE escogida permite diseñar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. Ofrece una serie de módulos independientemente de la del

modelado del negocio y sistema que sirven para todo el proceso de formalidad entre el equipo de desarrollo y el cliente.

Una vez que se expuso toda la fundamentación teórica, se define el *capítulo 2* como una propuesta del sistema que se desea desarrollar, en la cual se verán inmersas todas las definiciones técnicas que se aplicarán en el desarrollo del software.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA.

Este capítulo abordará la descripción del análisis y diseño llevada a cabo en la elaboración del sistema de actualizaciones automáticas para las oficinas registrales de SAREN.

2.1 Requisitos.

En esta sección se describen los requisitos del sistema de actualizaciones automáticas expresados en lenguaje natural. Pueden ser organizados por características pero también son apropiados métodos alternativos de organización como por ejemplo por usuario o subsistema.

2.1.1 Requisitos Funcionales.

RF1. Publicar los archivos actualizadores.

Los archivos actualizadores serán publicados en un FTP para ser descargados desde los servidores de cada una de las oficinas de SAREN.

RF2. Reconocer cuando debe efectuarse una actualización.

La aplicación debe ser capaz de reconocer si debe realizar el proceso de actualización de los subsistemas registrales.

RF3. Descargar la actualización.

La aplicación debe ser capaz de descargar los archivos actualizadores que se encuentren publicados en el FTP.

RF4. Hacer efectiva la actualización de los datos.

La aplicación debe ser capaz de reemplazar los archivos viejos por los nuevos descargados.

2.1.2 Requisitos No Funcionales.

Requisitos de Usabilidad.

- **Robustez.**

Está relacionado con la capacidad de recuperación de la información. El sistema contará con un control de las trazas y situaciones excepcionales que ocurran durante su correcta ejecución, de esta manera se podrá retomar la descarga de las actualizaciones a partir del punto de donde ocurrió el fallo.

- **Accesibilidad.**

Los archivos necesarios para actualizar los diferentes subsistemas estarán disponibles para cada una de las oficinas.

Requisitos de Fiabilidad.

- **Disponibilidad.**

El sistema estará en ejecución durante las 24 horas del día, efectuándose una actualización siempre que se detecte diferencias entre los archivos de los subsistemas y los archivos actualizadores. En caso de ser necesaria una actualización, el software la realizará automáticamente, sin intervención de ningún personal de las oficinas.

2.2 Arquitectura del Software.

Una definición reconocida de Arquitectura de Software es la brindada por Clements:"La Arquitectura de Software es una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones." (Universidad Tecnológica de Izúcar de Matamoros, 2010)

Otras de las definiciones de arquitectura de software la brinda el documento de la IEEE Std 1471-2000 que plantea: “La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución”. Esta es la definición que se ha tomado como oficial por todas las instituciones que desarrollan software a nivel mundial, como ha sido por ejemplo Microsoft. (DesarrolloWeb.com, 2001)

Existen diversos estilos arquitectónicos los cuales son clasificados en: Procesos Distribuidos, Sistemas Basados en Flujos de Datos, Sistemas de Llamada y Retorno (Call/Return), Componentes Independientes, Maquinas Virtuales, Sistemas de control, Sistemas Centrados en Datos, Programa Principal/Subrutinas, Sistemas de Transición de Estados y Arquitecturas Heterogéneas. (Cuesta, 2007)

Estilo Basado en Capas.

Se basa en la organización jerárquica de capas, donde cada capa provee servicios a la capa superior y es servido por la capa inferior. Los componentes definidos en este estilo son cada una de las capas. Los conectores son los protocolos de interacción entre las capas. La interacción está limitada a las capas adyacentes.

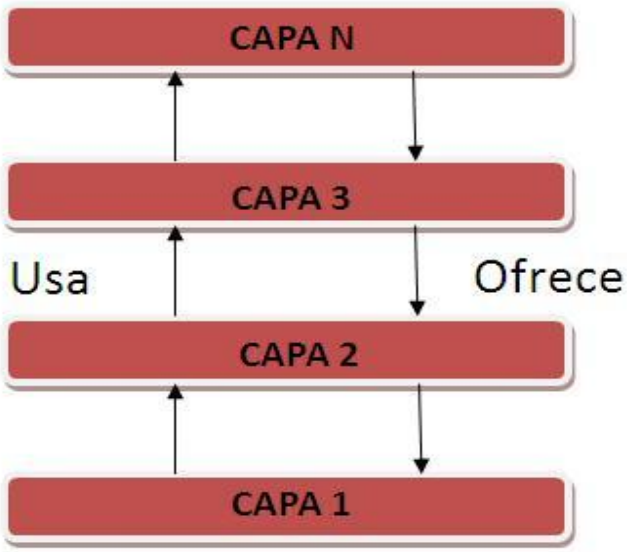

Figura 1. Estilo en Capas.

Las ventajas de este estilo, es que facilita la descomposición del problema en varios niveles de abstracción. Soporta fácilmente la evolución del sistema y los cambios solo afectan a las capas vecinas. Se pueden cambiar las implementaciones respetando las interfaces de comunicación existente con las capas adyacentes. El mayor inconveniente del estilo es que no todos los sistemas pueden estructurarse en capa.

Estilo Máquina Virtual o Intérprete.

Este estilo está formado por cuatro (4) componentes: un motor de simulación o interpretación, una memoria que contiene el código a interpretar, una representación del estado de la interpretación y una representación del estado del programa que se está simulando.

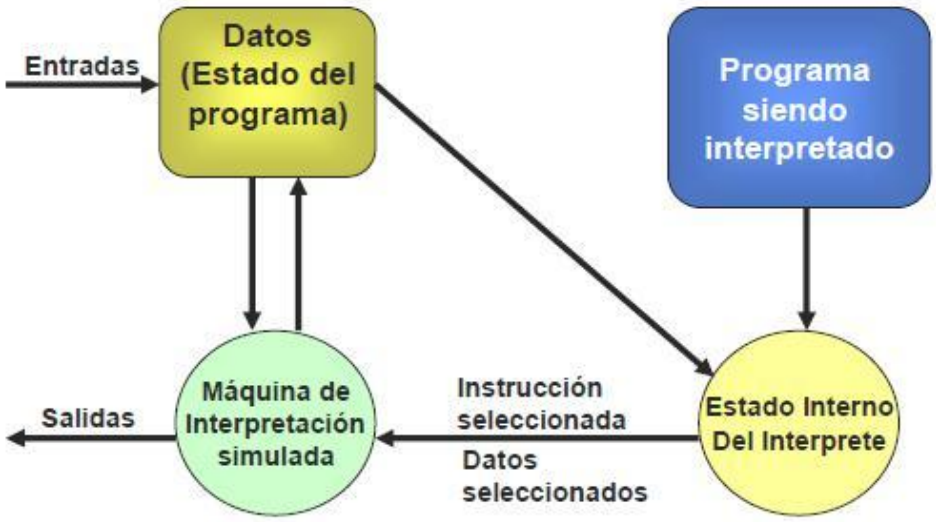

Figura 2. Estilo Maquina Virtual.

Las ventajas de este estilo son fundamentalmente en la solución de software para la solución de problemas en hardware. La desventaja que presenta el estilo de Maquina Virtual es que no siempre es aplicable.

Estilo Basado en Componentes.

Este estilo describe un acercamiento al diseño de sistemas como un conjunto de componentes que exponen interfaces bien definidas y que colaboran entre sí para resolver el problema.

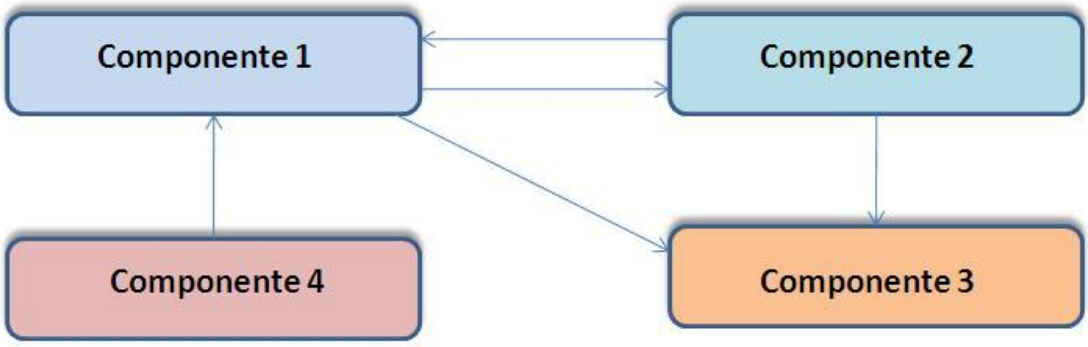

Figura 3. Estilo Basado en Componentes.

Algunas de las facilidades que brinda este estilo es que permite un fácil despliegue, ya que se puede sustituir un componente por su nueva versión sin afectar a otros componentes o al sistema. Permite además la reducción de costes, pues se pueden utilizar los componentes de terceros para abaratar los costes de desarrollo y mantenimiento. Es un estilo que independientemente del contexto los componentes pueden ser reutilizados en otras aplicaciones y sistemas.

Es recomendable el uso de este estilo cuando ya existen los componentes que son útiles para el sistema o aplicación que se vaya a desarrollar.

Estilo Cliente/Servidor.

La solución brindada está basada en la arquitectura cliente/servidor, la cual es una arquitectura de procesamiento cooperativo, donde uno de los componentes pide servicios a otro cuyo procesamiento de datos es de índole colaborativo. (Biblioteca Virtual Católica del Norte, 2005)

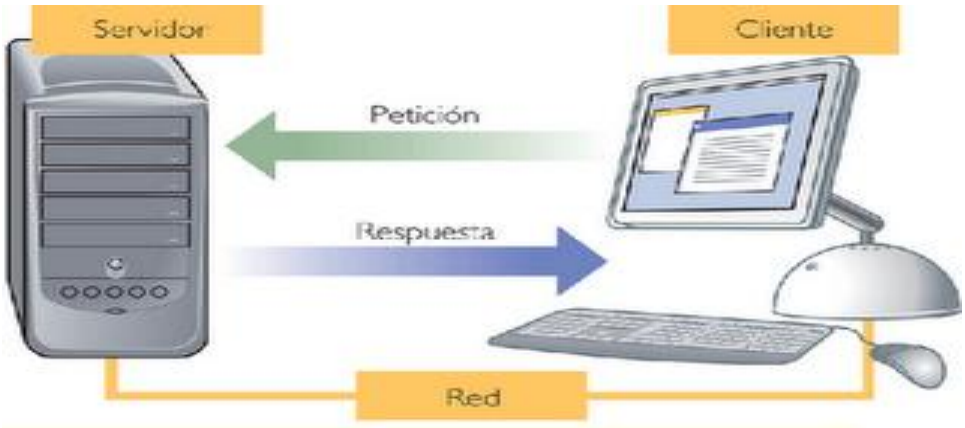

Figura 4. Estilo Cliente/Servidor.

El modelo cliente/servidor proporciona al usuario final el acceso transparente a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo en múltiples plataformas. El modelo soporta un ambiente distribuido en el cual los requerimientos de servicios son hechos por las estaciones de trabajo clientes.

Este estilo garantiza que el servidor no realice ninguna petición al cliente. Arma en alguna medida de una seguridad para los datos, ya que estos se almacenan en el servidor que generalmente ofrece más control sobre la seguridad.

2.2.1 Patrones.

Un patrón es una solución genérica que se brinda a un tipo de problema ya conocido y solucionado anteriormente. Existen muchos patrones los cuales se clasifican en categorías entre ellos: patrones de arquitectura y patrones de diseño.

Patrones de Arquitectura.

Modelo-Vista-Controlador.

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes: el *Modelo* administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de

información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

La *Vista*, maneja la visualización de la información; el *Controlador* interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. (Carlos Reynoso, Nicolás Kiccillof, 2004)

Tuberías y Filtros.

Este es un estilo de flujo de datos. Este que conecta componentes computacionales (filtros) a través de conectores (tuberías), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. (Carlos Reynoso, Nicolás Kiccillof, 2004)

Patrones de Diseño.

Dentro de los patrones de diseños se encuentran los patrones GRASP[18] que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. El nombre se surge para indicar la importancia de captar estos principios. (Larman, 1999)

Experto.

Este patrón asigna una responsabilidad al experto de información: la clase que cuenta con la información necesaria para cumplir las responsabilidades asignadas a ella. Plantea básicamente que los objetos realizan las operaciones relacionadas con la información que poseen. Esto soporta un bajo acoplamiento. Este patrón propicia además que las operaciones se distribuyan entre las clases que cuentan con la información requerida para desarrollarla, permitiendo así definiciones de clases sencillas y más cohesivas. (Larman, 1999)

[18] General Responsibility Assignment Software Patterns.

Bajo Acoplamiento.

Este patrón permite asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. El bajo acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios. (Larman, 1999)

Alta Cohesión.

Este patrón permite mantener un control en la complejidad de las clases, manteniendo un número relativamente pequeño de operaciones. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Este patrón mejora la calidad y facilidad con que se puede entender el diseño, se genera un bajo acoplamiento y permite fomentar la reutilización. (Larman, 1999)

2.3 Definición general de los casos de uso.

Los casos de uso describen una secuencia operaciones realizadas en el negocio que culmina con un resultado de valor observable para un actor individual del negocio. (Universidad Central San Carlos de Guatemala, 1993)

2.3.1 Identificación de los actores

Los actores son personas, grupos, máquinas u organizaciones con las cuales el negocio interactúa, y representan el rol que se juega cuando se interactúa con el negocio para beneficiarse de los resultados. Debido a la sencillez del negocio en cuestión solo definimos un actor, el cual se describe a continuación.

2.3.2 Identificación y descripción de los casos de uso

En este epígrafe se hará una descripción textual de los casos de uso del sistema usando un formato de documento que se emplea en la especificación de casos de uso del negocio.

Caso de uso		
Nombre	Publicar Archivos de Actualización.	
Propósito	Publicar la actualización que serán descargadas por los servidores de las oficinas.	
Actores: Administrador del Centro de Datos.		
Resumen: El administrador selecciona los ficheros a descargar y los publica en un servidor.		
Flujo Normal de los Eventos		
Acción del actor		**Respuesta del sistema**
1- El Administrador selecciona los ficheros a cargar.		2- Genera un fichero XML con datos necesarios para llevar a cabo la actualización. 3- Publica el fichero XML y los ficheros actualizadores en un servidor.
Poscondiciones	Se publican los ficheros actualizadores y el archivo XML.	

Tabla 2. Descripción del caso de uso Publicar Archivos de Actualización.

Caso de uso		
Nombre	Actualizar Aplicación.	
Propósito	Realizar la descarga de los ficheros para actualizar la aplicación en ejecución en la terminal.	
Actores: Sistema de Actualización Automática.		
Resumen: El sistema descarga los archivos del servidor donde están publicados y actualiza la aplicación en cuestión.		
Flujo Normal de los Eventos		
Acción del actor		**Respuesta del sistema**

1- Descarga el archivo XML con la configuración de la actualización.	2- Verifica las fechas y la versión del archivo XML. 3- Descarga los archivos en una carpeta temporal. 4- Sobrescribe los archivos de la aplicación por los archivos descargados.
Poscondiciones	Se realiza la actualización de la aplicación.
Flujo Alternativo	
	3- Si la versión del archivo XML coinciden no se realiza ninguna descarga.
	3.1- Si ocurre algún error de conexión durante la descarga de los archivos, el sistema lo registra y se mantiene en espera. 4- El sistema intenta descargar los archivos nuevamente para la carpeta temporal en el tiempo de actualización establecido. 5- Sobrescribe los archivos de la aplicación por los archivos descargados.

Tabla 3. Descripción del caso de uso Actualizar Aplicación.

2.4 Diagrama de casos de uso del negocio.

Muestra gráficamente la relación entre los actores y los casos de uso del negocio, especificando que actores inician que casos de usos así como la relación existente entre los diferentes casos de usos identificados.

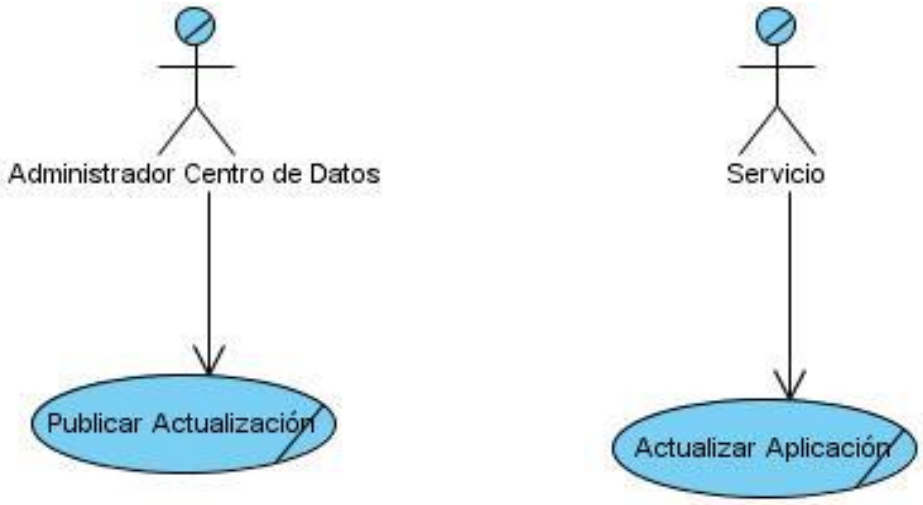

Figura 5. Diagrama de Casos de uso del negocio.

2.5 Conclusiones.

En este capítulo se establecieron los requisitos funcionales y no funcionales con los cuales la aplicación debe cumplir para satisfacer las demandas del cliente. Se definió la arquitectura cliente/servidor pues brinda soporte a los requisitos captados y permite la ejecución de la actualización, siendo esta transparente al usuario.

Se definieron y describieron los casos de uso del negocio que serán automatizados, especificando la relación existente entre los diferentes casos de uso identificados y los actores que los inician, logrando así corregir los errores y fallas que hasta ahora vienen afectando al sistema de actualización automática en explotación.

Al tenerse definido los requisitos con los que debe cumplir la aplicación, la arquitectura a utilizar y los casos de uso del negocio, se da paso al capítulo 3 que aborda el diseño e implementación de la propuesta del sistema de actualizaciones automáticas.

CAPÍTULO 3: DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE ACTUALIZACIONES AUTOMÁTICAS.

El objetivo de este capítulo es desarrollar los artefactos más importantes para la actividad de diseño, partiendo de la fundamentación del tema y el estudio del arte de las diferentes herramientas, metodologías y tecnologías existentes. El punto de partida para la elaboración de este diseño fueron las especificaciones de los casos de uso del negocio, los requisitos asociados a estos y las reglas generales del negocio. Este análisis permite detallar las características funcionales del software así como algunas restricciones que deba cumplir el sistema.

3.1 Diagrama de clases del diseño del sistema.

Estos diagramas muestran de forma detallada las clases con sus atributos y métodos que permitirán que se realicen todas las operaciones y como están relacionadas las clases entre ellas.

Diagrama de Clases GeneradorXML.

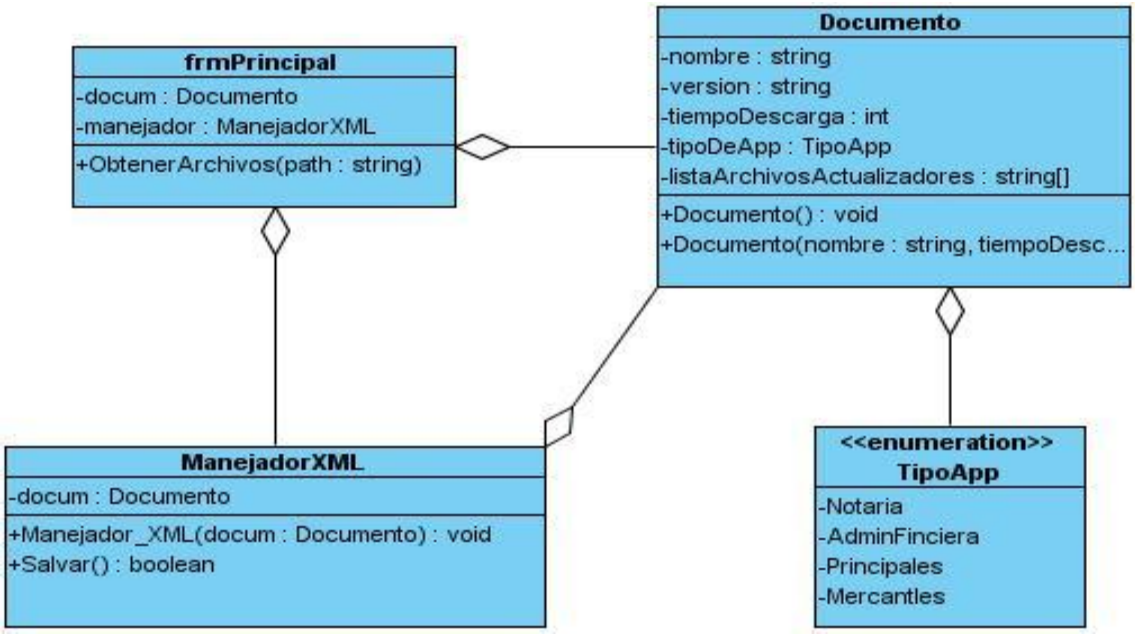

Figura 6. Diagrama de Clases GeneradorXML.

.

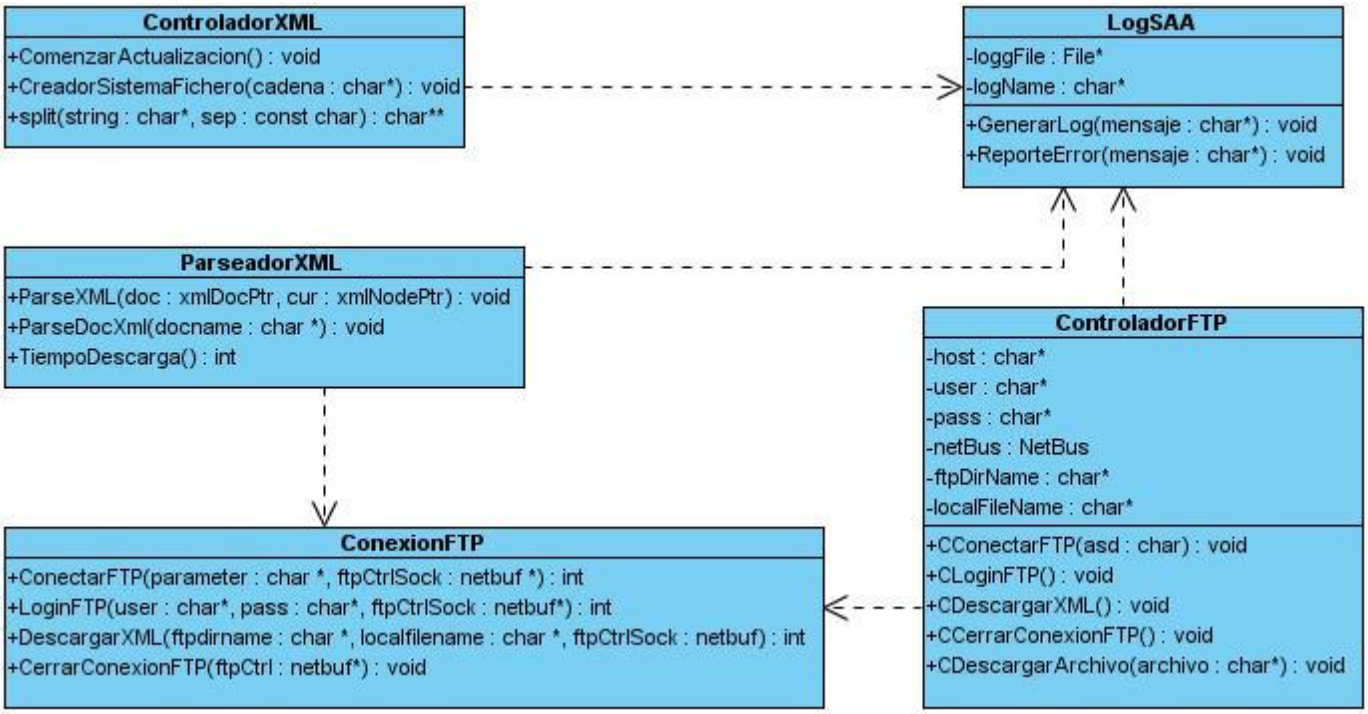

Figura 7. Diagrama de Clases ActualizadorSAA.

3.2 Implementación del sistema.

Para proceder a la implementación del sistema una vez creados los diagramas de diseño, se procede a la definición del estándar de codificación utilizándose el estilo Camelcase para la declaración de variables, atributos, parámetros y componentes y el estilo Pascal para nombrar clases y métodos. (**Ver Anexo6)**

En el desarrollo del software GeneradorXML se utilizaron las librerías *QtCore*, *QtGui* y *QtXml.* En el diseño de la interfaz se hizo uso de la herramienta *Qt 4 Designer*. Para el software ActualizadorSAA se hace necesario el uso de las librerías *Ftplib* y *Libxml2*, desarrollándose un *daemon* en el lenguaje *C*.

3.3 Diagrama de componentes.

Este diagrama de componentes muestra la relación de dependencia existente entre los componentes de la solución implementada.

Diagrama de componentes.

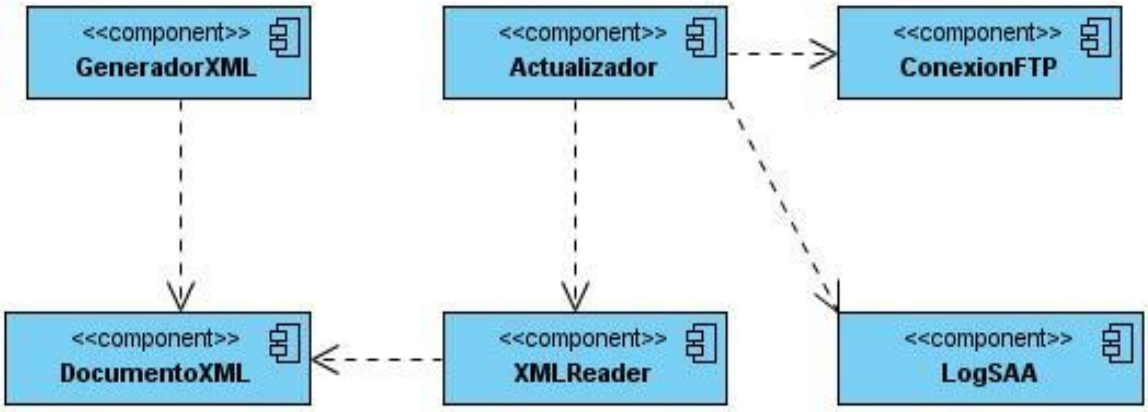

Figura 8. Diagrama de componentes.

El siguiente epígrafe expone mediante un diagrama como quedarán distribuidos los elementos de hardware para desplegar el software.

3.4 Diagrama de despliegue.

El diagrama de despliegue hace una representación gráfica aproximada de cómo quedarán ubicados los elementos de hardware para poner a funcionar el sistema de actualizaciones automáticas.

Diagrama de despliegue.

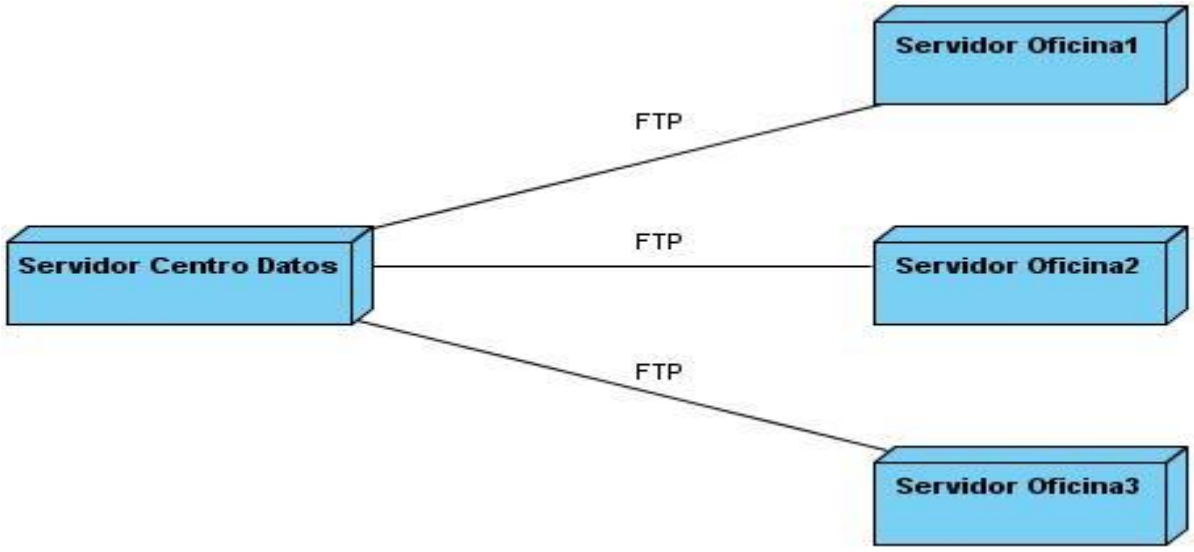

Figura 9. Diagrama de despliegue.

Se pueden notar que están representados los servidores, uno en el Centro de Datos y los servidores de cada una de las oficinas. El software ActualizadorSAA se ejecuta en el servidor del las oficinas y descarga las actualizaciones publicadas en el Centro de Datos mediante el protocolo FTP.

Para el despliegue de este sistema de actualizaciones automáticas no se requiere de nuevos elementos de hardware o modificación en la topología de la red privada de la cual hace uso SAREN, por lo que su puesta en funcionamiento no requiere de inversiones adicionales.

3.5 Conclusiones.

Se diseñaron e implementaron dos software: el GeneradorXML, el cual es de interfaz gráfica, utilizado por el administrador del Centro de Datos para publicar las actualizaciones y generar el archivo XML y el ActualizadorSAA que es un demonio que se ejecutará en los servidores de las oficinas de forma automática y totalmente transparente para los usuarios finales. Este demonio efectuará las descargas de los archivos actualizadores y hará efectiva dicha actualización. Además lleva un control de cada evento ocurrido durante la ejecución del proceso de actualización y en caso de que exista alguna falla es capaz de interrumpir la descarga para luego ser retomada a partir del último archivo descargado correctamente.

Luego de culminada la fase de diseño e implementación se procede a realizar las pruebas al sistema propuesto, con el objetivo de verificar el cumplimiento de los requisitos captados y la calidad del software.

CAPÍTULO 4: VALIDACIÓN Y PRUEBAS.

La etapa de pruebas tiene como objetivo definir el alcance, recursos requeridos y responsables de identificación de riesgos en este proceso. El propósito de esta etapa es establecer la línea base por la cual el equipo de calidad se regirá para realizar pruebas en el tiempo determinado, estas tendrán como resultado la validación de los requisitos.

4.1 Métricas del Diseño

Las métricas orientadas al tamaño se centran en el recuento de atributos y operaciones para cada clase individual y los valores promedio para el sistema como un todo. (Pressman, 2008)

La métrica tamaño de clase establece que el tamaño general de una clase puede medirse a través de la medida del número total de operaciones y atributos (heredados y privados de la interfaz), que se encapsulan dentro de la clase. Los valores grandes para esta métrica, indican que la clase debe tener bastante responsabilidad. Esto reducirá la reutilización de esta clase y complicará la implementación y las pruebas. Los promedios del número de atributos y operaciones de las clases se pueden calcular de manera que mientras menor sea el valor promedio, mayor será la posibilidad de reutilizar la clase. (Pressman, 2008)

A continuación se muestra la relación de los intervalos de valores para determinar si el valor de TC[19] se considera pequeño, medio o grande.

Umbrales para la Métrica TC.

Umbral	TC (Total de Atributos y Operaciones)
Menor o igual que 20 (TC <= 20)	Pequeño
Entre 20 y 30 (20 < TC <= 30)	Medio
Mayor que 30 (TC > 30)	Grande

Tabla 4. Umbrales para la Métrica Tamaño de Clases.

[19] Tamaño de clases.

La métrica Relaciones entre Clases (RC) está dada por la cantidad de relaciones de uso que existe entre las distintas clases que forman el diseño propuesto. Se aplica a las mismas clases en las que fue aplicada la métrica TC. Los aspectos de calidad que se miden son: Acoplamiento, Complejidad de mantenimiento y Reutilización.

Cantidad de Relaciones entre clases.

Subsistema	Clases	Cantidad de Relaciones de Uso
GeneradorXML	frmPrincipal.cpp	2
GeneradorXML	Documento.cpp	0
GeneradorXML	ManejadorXML.cpp	1
ActualizadorSAA	ControladorFTP.c	2
ActualizadorSAA	ConexionFTP.c	1
ActualizadorSAA	LogSAA.c	3
ActualizadorSAA	ParseadorXML.c	1
ActualizadorSAA	ControladorXML.c	2

Tabla 5. Cantidad de Relaciones.

Acoplamiento entre clases.

Categoría	Relaciones de uso	Cantidad de clases
Ninguno	0	2
Bajo	1	4
Medio	2	2
Alto	3 o más	0

Tabla 6. Acoplamiento.

Reutilización de las clases.

Categoría	Criterio	Cantidad de Clases
Baja	>2* Promedio.	0
Media	> Promedio. Y <=2* Promedio.	2
Alta	<= Promedio.	6

Tabla 7. Reutilización.

Complejidad de Mantenimiento de clases.

Categoría	Criterio	Cantidad de clases
Baja	<=Promedio.	6
Media	> Promedio. Y <=2* Promedio.	2
Alta	>2* Promedio.	0

Tabla 8. Complejidad de Mantenimiento.

4.2 Casos de pruebas del sistema.

Las pruebas que se llevan a cabo son de Caja Negra para comprobar que el sistema de actualizaciones automáticas cumpla con las funcionalidades requeridas por el cliente. Los requisitos a probar serán: la capacidad de publicar los archivos actualizadores del sistema, reconocer cuando debe efectuarse la actualización, descargar la actualización y hacerla efectiva. Estas pruebas responderán a los requisitos funcionales del sistema. Para dar cumplimiento a los requisitos no funcionales se someterá a prueba la robustez del sistema. Los requisitos no funcionales: accesibilidad y disponibilidad no serán sometidos a pruebas, pues estos requisitos los debe cumplir el servidor donde se encuentre instalado el sistema de actualizaciones automáticas.

Caso de Prueba de Aceptación al GeneradorXML	
Código Caso de Prueba: RF-1.1	**Nombre de Prueba:** Copiar actualizaciones.
Nombre del Probador: Angel Alberto Bello Caballero.	
Descripción de la Prueba: Se verificará que el sistema copie correctamente los archivos actualizadores a la ubicación especificada.	
Condiciones de Ejecución: Que exista ubicación de origen y destino de los archivos actualizadores.	
Entrada / Pasos de ejecución: Se invoca el evento *clicked* del botón *Generar*.	
Resultado Esperado: Que se copien los archivos actualizadores a la ubicación especificada.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 9. Caso de Prueba de Aceptación: Copiar Actualizaciones.

Caso de Prueba de Aceptación al GeneradorXML	
Código Caso de Prueba: RF-1.2	**Nombre de Prueba:** Generar XML.
Nombre del Probador: Angel Alberto Bello Caballero.	
Descripción de la Prueba: Se verificará que el sistema genere correctamente el archivo XML con los datos necesarios para realizar la actualización.	
Condiciones de Ejecución: Que se introduzcan los datos necesarios para la correcta generación del archivo XML.	
Entrada / Pasos de ejecución: Se invoca el método *Salvar* de la clase *ManejadorXML*.	
Resultado Esperado: Que se genere el archivo XML en la ubicación especificada.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 10. Caso de Prueba de Aceptación: Generar XML.

Caso de Prueba de Aceptación al ActualizadorSAA	
Código Caso de Prueba: RF-2.1	**Nombre de Prueba:** Reconocer la existencia de las actualizaciones.
Nombre del Probador: Manuel Alejandro Diaz Alonso.	
Descripción de la Prueba: Que el software reconozca si se debe realizar una actualización.	
Condiciones de Ejecución: Que se encuentre publicado el archivo XML en el Centro de Datos.	
Entrada / Pasos de ejecución: Se descarga el archivo XML y se comparan las versiones de archivo relacionado a la última actualización y el archivo descargado.	
Resultado Esperado: Que el daemon reconozca si debe realizarse la actualización y la ejecute.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 11. Caso de Prueba de Aceptación: Reconocer la existencia de las actualizaciones.

Caso de Prueba de Aceptación al ActualizadorSAA	
Código Caso de Prueba: RF-3.1	**Nombre de Prueba:** Descargar la actualización.
Nombre del Probador: Manuel Alejandro Diaz Alonso.	
Descripción de la Prueba: Se descargan los archivos del servidor del Centro de Datos.	
Condiciones de Ejecución: Que se haya publicado una nueva actualización.	
Entrada / Pasos de ejecución: Se invoca el método *DescargarArchivo* de la clase *ControladorFTP*.	
Resultado Esperado: Que se descarguen los archivos publicados en el servidor.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 12. Caso de Prueba de Aceptación: Descargar la actualización.

Caso de Prueba de Aceptación al ActualizadorSAA	
Código Caso de Prueba: RF-4.1	**Nombre de Prueba:** Hacer efectiva la actualización de los datos.
Nombre del Probador: Manuel Alejandro Diaz Alonso	
Descripción de la Prueba: Se reemplazan los archivos viejos por los descargados.	
Condiciones de Ejecución: Que se haya completado la descarga de archivos.	
Entrada / Pasos de ejecución: Se invoca el método *CopiarArchivo* de la clase *ControladorXML*.	
Resultado Esperado: Que se sobrescriban los archivos viejos por los archivos descargados.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 13. Caso de Prueba de Aceptación: Hacer efectiva la actualización de los datos.

Caso de Prueba de Aceptación al ActualizadorSAA	
Código Caso de Prueba: RNF-1.1	**Nombre de Prueba:** Recuperación ante situaciones excepcionales.
Nombre del Probador: Manuel Alejandro Diaz Alonso.	
Descripción de la Prueba: Ante la ocurrencia de una situación excepcional durante la descarga de los archivos, el sistema debe ser capaz de reanudar la ejecución de la misma desde el último archivo descargado correctamente.	
Condiciones de Ejecución: Que se esté realizando una actualización y ocurra una situación excepcional.	
Entrada / Pasos de ejecución: Se invoca el método *sleep* en la clase *ParseadorXML.*	
Resultado Esperado: Que se continúe con la actualización una vez rebasada la situación excepcional.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 14. Caso de Prueba de Aceptación: Recuperación ante situaciones excepcionales.

Caso de Prueba de Aceptación al ActualizadorSAA	
Código Caso de Prueba: RNF-1.2	**Nombre de Prueba:** Control y Traza
Nombre del Probador: Manuel Alejandro Diaz Alonso	
Descripción de la Prueba: Se generan 2 archivos, uno para el reporte de errores y otro para el reporte de todos los eventos ocurridos.	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución: Se invoca el método *GenerarLog* o *ReportarError* de la clase *LogSAA*.	
Resultado Esperado: Que se registren todos los eventos y errores ocurridos.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 15. Caso de Prueba de Aceptación: Control y Traza.

Para la prevención de entrada de datos erróneos por parte del usuario se realizó la validación de cada uno de los campos de la interfaz del software GenerarXML. Se validó que para que se pueda generar una actualización no se debe dejar vacío ningún campo, restringiéndose la entrada de números

enteros en el campo *Tiempo de Descarga*. Además se verificó que en la ubicación de origen debe existir al menos un archivo actualizador. (**Ver Anexos 1 al 5**).

4.3 Conclusiones.

En este capítulo se diseñaron y aplicaron métricas al diseño y una serie de casos de pruebas para la validación del sistema propuesto, las cuales arrojaron resultados satisfactorios. Una vez aplicada la métrica TC y analizados los resultados obtenidos se puede concluir que el mayor por ciento de las clases se clasifican de tamaño pequeño, por lo que se considera un resultado positivo para el software según los parámetros establecidos en la métrica. Esto indica que las clases del sistema son reusables y fáciles de comprobar.

La aplicación de la métrica RC arrojó resultados positivos. El acoplamiento existente entre las clases es moderado, es decir, el necesario para contribuir a una alta cohesión, el 25% no tiene ningún acoplamiento, el 50% es bajo y el 25% es medio. El nivel de reutilización de las clases es bastante bueno, el 75% de las clases pueden ser altamente reutilizables. Sucede de la misma manera con la complejidad de mantenimiento ya que el 75% de las clases son fáciles de reparar.

La evaluación de los casos de pruebas resulto un 100% satisfactorio, esto muestra que el producto responde los requisitos planteados por el cliente, evidenciando así que el software fue diseñado e implementado con alta calidad.

CONCLUSIONES GENERALES

Se realizó un estudio de los estilos arquitectónicos, patrones y herramientas más convenientes para concretar el desarrollo de un sistema de actualizaciones automáticas. Se modelaron los procesos de negocio que comprenden un sistema de actualizaciones automáticas para la comprensión de las funcionalidades necesarias para su correcto funcionamiento.

Como resultado del diseño e implementación del sistema propuesto, se obtuvo la construcción de un producto informático. El sistema desarrollado responde a las peticiones del cliente, solucionando el manejo de situaciones excepcionales mediante un control de cada evento que ocurra durante su ejecución y los procesos de actualización y fallas sin la mínima intervención del usuario final o personal técnico. Además permite la ejecución de despliegues centralizados requiriendo de menor cantidad de personal para realizar esta operación.

Para la validación en cuanto a calidad y eficiencia en el proceso de actualizaciones automáticas se le aplicaron métricas al diseño y pruebas de Caja Negra al software para comprobar el funcionamiento de las funcionalidades. En esta etapa se obtuvo como resultado que el producto tiene la calidad requerida para su implantación.

Se desarrolló un servicio de actualizaciones automáticas mediante el cual se logra mayor eficiencia en el control de las conexiones y descargas de archivos y contribuye así a un mejor aprovechamiento del tiempo y recursos de las entidades adscriptas a SAREN.

Este trabajo sirve de guía para futuros sistemas de actualizaciones automáticas ya sean independientes o como parte de un software, sentando las bases para que una vez creada la infraestructura en Cuba se desarrollen software que sean capaces de actualizarse automáticamente para mantener al usuario con la última versión de su aplicación.

RECOMENDACIONES

Elaborar un manual de usuario para la correcta comprensión del funcionamiento del sistema de actualizaciones automáticas propuesto.

Realizar un proceso de cifrado a los archivos que serán descargados para una vez realizada su descarga comprobar que no han sido alterados.

Realizar pruebas piloto al sistema propuesto en un ambiente controlado para comprobar su ejecución.

Implementar una funcionalidad que permita visualizar desde el Centro de Datos el estado de las actualizaciones en cada una de las oficinas.

Extender el sistema de actualizaciones automáticas hacia los terminales clientes donde se encuentran ejecutándose cada uno de los subsistemas de SAREN.

BIBLIOGRAFIA

Application Architecture for .NET:Designing Applications and Services. (2002). Microsoft Corporation.

Becerra Guzman, S. L. (2009). *Desarrollo de un sistema de vigilancia corporativo compatible con dispositivos de telefonia movil.*

Biblioteca Virtual Católica del Norte. (2005). Retrieved 1 15, 2011, from http://biblioteca.ucn.edu.co/repositorio/Ingenieria/RedesDatosYConectividadI/documentos/Arquitectura-cliente-servidor.pdf

Booch, G. (2006). *On Architecture.*

Bosch, J. (2000). *Design and Use ofSoftware Architectures.*

Camacho, I. (2009). *Administración de Proyectos de Software.* Cochabamba, Bolivia: Universidad Mayor de San Simón.

Carlos Reynoso, Nicolás Kiccillof. (2004). *Estilos y Patrones en la Estrategia de Microsoft.*

Centro de Estudios para el Perfeccionamiento. (2009). Retrieved 12 5, 2010, from http://cepes.uh.cu/investigaciones.html

Conocimientos Web. (2010). Retrieved 12 09, 2010, from http://www.conocimientosweb.net/dcmt/downloads-cat-105.html

Cuesta, C. E. (2007). Arquitectura de Software. Universidad Rey Juan Carlos .

DesarrolloWeb.com. (2001, 7 14). Retrieved 12 5, 2010, from http://www.desarrolloweb.com/articulos

Doc.ubuntu-es. (2010, 3 20). Retrieved 12 15, 2010, from http://doc.ubuntu-es.org/Notas_sobre_actualizaciones

Escribano, G. F. (2002). *Introducción a Extreme Programming.*

eumet.net. (2009). Retrieved 12 15, 2010, from http://www.eumed.net/libros/2009c/584/Por%20que%20utilizar%20Scrum%20para%20desarrollar%20aplicaciones%20web.htm

Facultad de Informatica. (2010). Retrieved 11 09, 2010, from http://www.fdi.ucm.es/profesor/lgarmend/FBD/Tema%202.3%20Algebra%20relacional%20v6.pdf

Figueroa, R. G. (2008). *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación.

Fleck, J. (2004). *Libxml Tutorial.*

Griffith, A. (2002). *GCC: The Complete Reference.*

Jalón, J. G. (1998). *Aprenda C++ como si estuviera en primero.*

L. Bass, P. Clements, R. Kazman. (2003). *Software Architecturein Practice(2nd Edition).*

Larman, C. (1999). *UML y Patrones: Introducción al análisis y diseño orientado a objeto.*

Lenguajes de Programacion. (2009). Retrieved 12 16, 2010, from http://www.lenguajes-de-programacion.com/programacion-orientada-a-objetos.shtml

M. Shaw, D. Garlan. (1997). *Software Architecture, Perspectives on an Emerging Discipline.*

Mellor, S. (2005). *UML Distilled: A Brief Guide to the Standard Object Modeling Language.* I E E E C o m p u t e r S o c i e t y.

Mercer, D. (2001). *Fundamentos de Programacion en XML.*

Microsoft Technet. (2010, 8). Retrieved 12 14, 2010, from http://technet.microsoft.com/es-es/library/cc781859%28WS.10%29.aspx

NetBeans. (2001). Retrieved 12 15, 2010, from http://netbeans.org/

P. Clements, F. Bachmann, L. Basset al. (2003). *DocumentingSoftware Architectures: Views and Beyond.*

Pfau, T. (1997). *FTP Library Routines Release 3.*

Piñero, L. J. (2009, 9 19). *Universidad del Salvador*. Retrieved 12 08, 2010, from http://www.salvador.edu.ar/fcecs/Noticia_Facultad/Noticias_Carrera_Periodismo/videoconferencia.htm

Pressman, R. S. (2008). *Ingeniería de Software. Un enfoque práctico.*

Ribao, J. M. (2006). *Soluciones open-source de interoperatividad entre java y cli.*

Schildt. (1995). *C++. Guía de Autoenseñanza.*

The GNU Compiler Collection. (2010). Retrieved 12 16, 2010, from http://gcc.gnu.org/

Universidad Central San Carlos de Guatemala. (1993). Retrieved 2 11, 2011, from http://biblioteca.usac.edu.gt/tesis/08/08_5524.pdf

Universidad de las Américas Puebla. (2010, 11 11). Retrieved 12 21, 2010, from http://social.udlap.mx/2010/11/actualizarse/

Universidad Tecnológica de Izúcar de Matamoros. (2010). Retrieved 1 12, 2011, from http://www.utim.edu.mx/~raycv/materias/univer/unidad_III/

Velthuis, P. (1996). *Calidad en el desarrollo y mantenimiento del software.*

Visual Paradigm. (2000). Retrieved 12 16, 2010, from http://www.visual-paradigm.com/

W3C. (2011). Retrieved 12 15, 2010, from http://www.w3.org/XML/

Watson, D. (2004, 5). Retrieved 12 08, 2010, from http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html

ANEXOS

Anexo 1. Documento XML generado.

En el anexo se muestra el documento XML generado correctamente con el software GeneradorXML.

```
−<Documento>
    <version>1</version>
    <nombre>prueba1.0</nombre>
    <tiempoDescargaMin>15</tiempoDescargaMin>
    <tipoDeApp>AdminFinciera</tipoDeApp>
  −<archivosActualizadores>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/Controles.csproj.vspscc
      </Archivo>
      <Archivo>Proyecto/Controles/ControlBuscarPersona/Cedula.cs</Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/ControlNumerico.resx
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/NroTramite.cs
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/AssemblyInfo.cs
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/RIFPlano.cs
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/TestPunto.cs
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/TestPunto.resx
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/NroTramite.resx
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/PuntoMillaresExpediente.cs
      </Archivo>
    −<Archivo>
        Proyecto/Controles/ControlBuscarPersona/BusquedaPersona.resx
      </Archivo>
```

Figura 10. Archivo XML.

Anexo 2. Prueba al software GeneradorXML.

En el anexo se muestra un mensaje de notificación en caso de que en la carpeta seleccionada como origen de las actualizaciones no existan archivos.

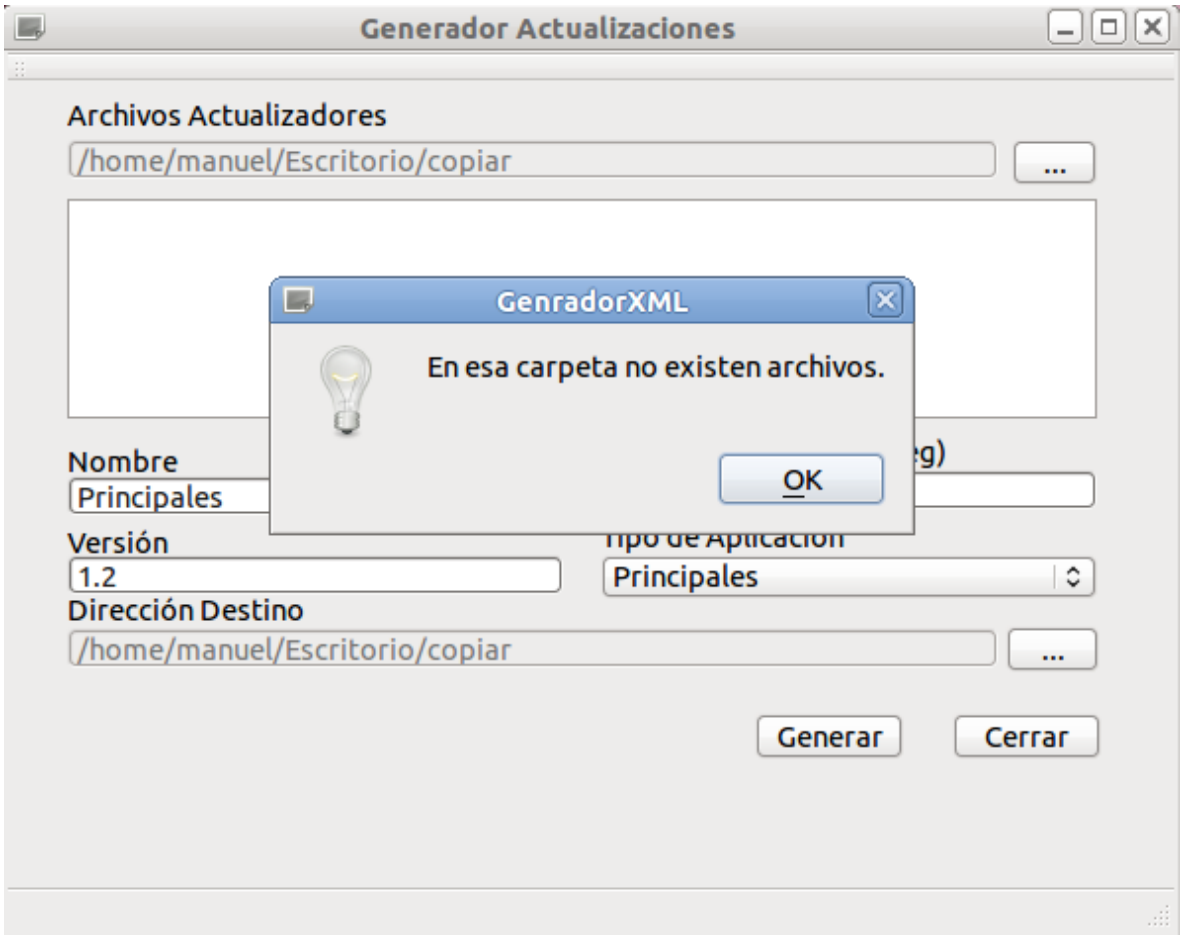

Figura 11. Validación para existencia de archivos actualizadores.

Anexo 3. Prueba al software GeneradorXML.

En el anexo se muestra un mensaje de notificación en caso de que exista algún campo sin el dato requerido.

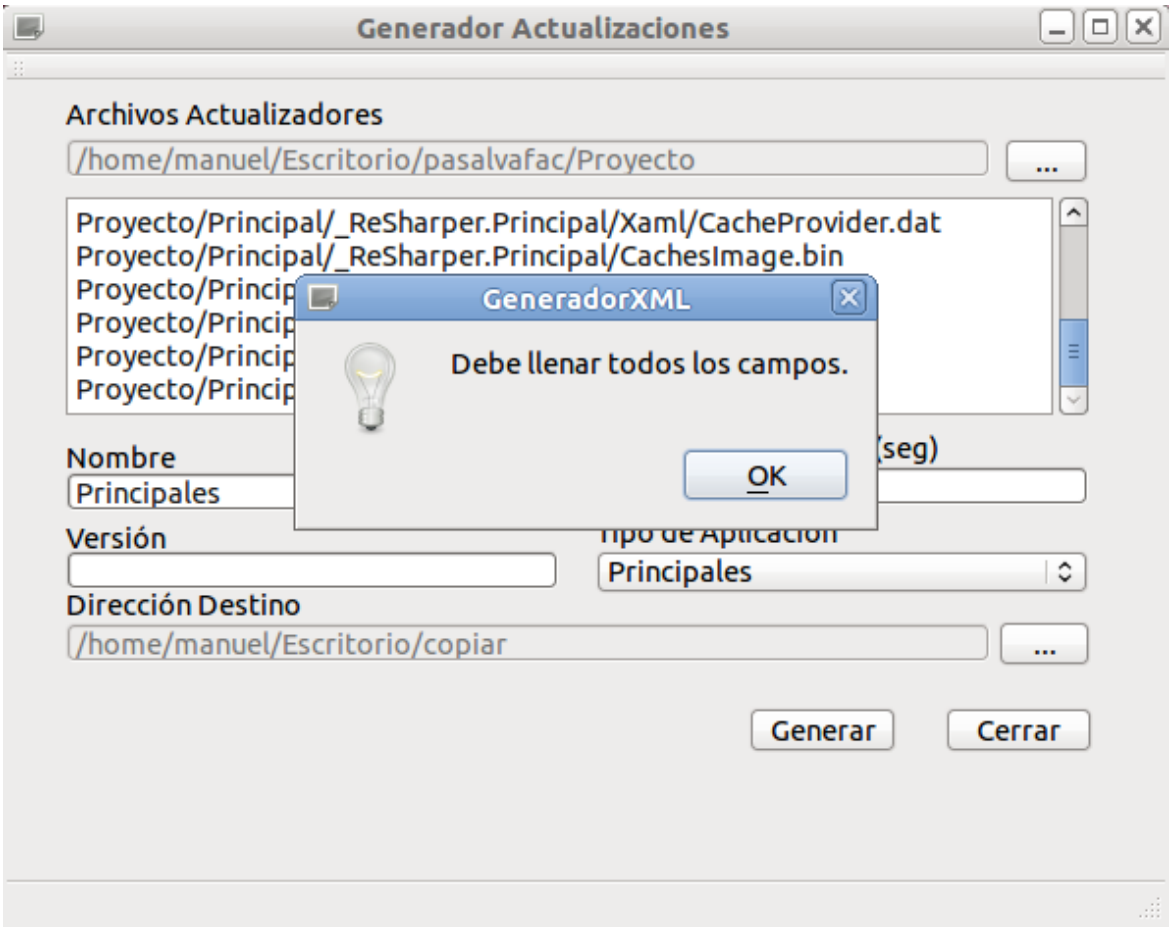

Figura 12. Validación para la entrada de Datos.

Anexo 4. Prueba al software GeneradorXML.

En el anexo se muestra un mensaje de notificación en caso de que se haya generado correctamente las actualizaciones.

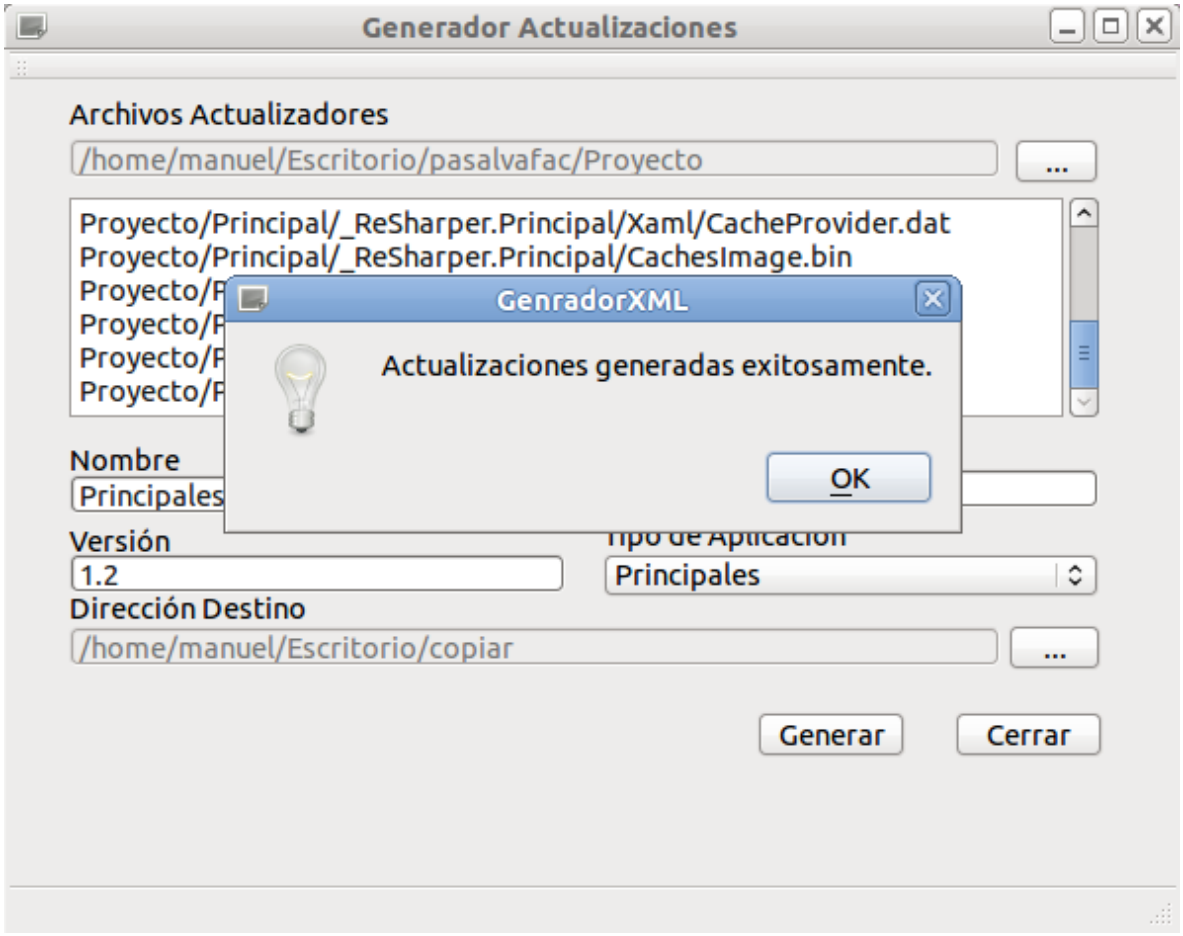

Figura 13. Generación exitosa de la actualización.

Anexo 5. Prueba al software GeneradorXML.

En el anexo se muestra la interfaz con los datos correctamente escritos.

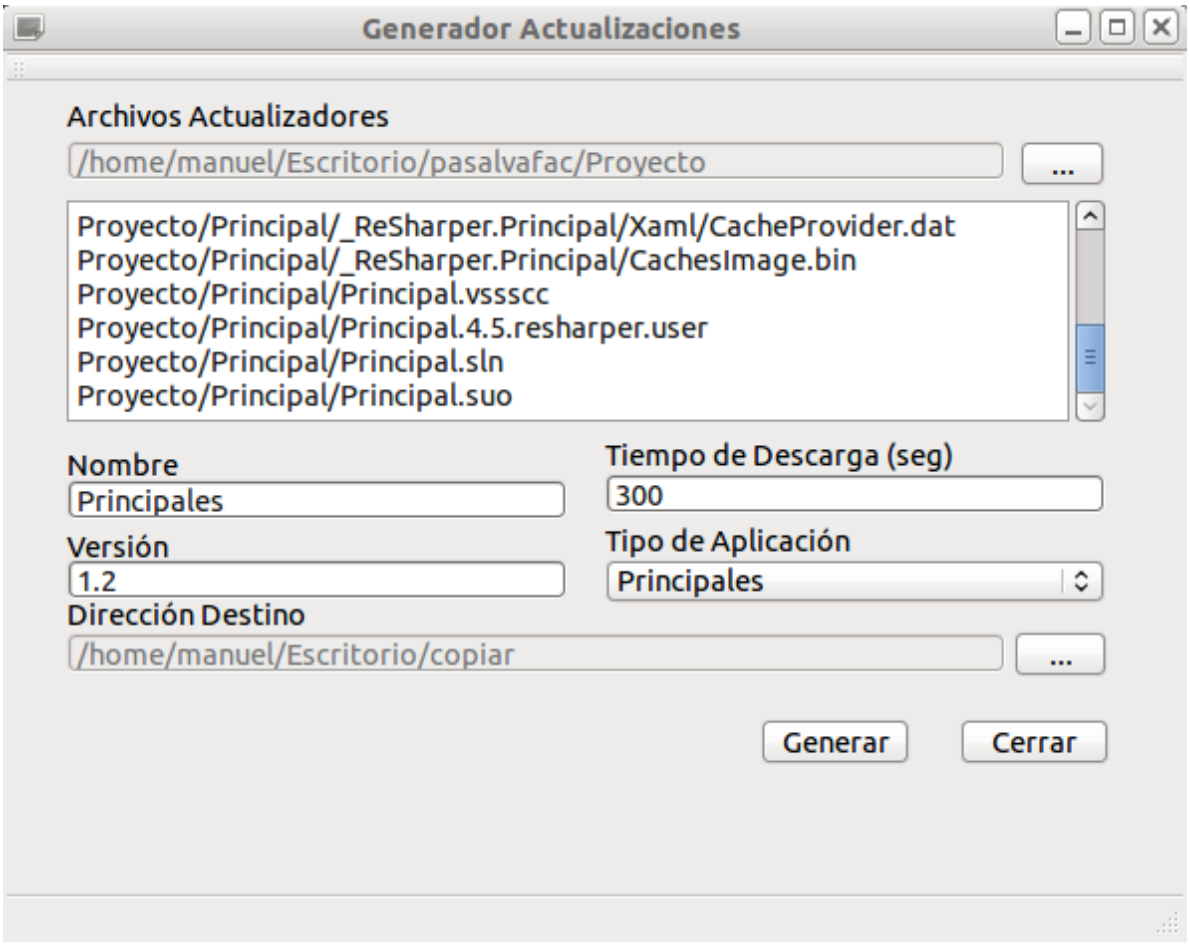

Figura 14. Datos correctamente escritos.

Anexo 6. Estilos y estándares de codificación.

En el anexo se muestra los estilos y estándares de codificación seguidos para implementar el sistema de actualizaciones automáticas propuesto.

Estilo de Pascal: Capitaliza la primera letra de cada palabra.

Ejemplo: Clase, ClaseCompuesta.

Estilo Camelcase: Capitaliza la primera letra de cada palabra, excepto para la primera palabra.

Ejemplo: variable, variableCompuesta.

Estándares de Codificación.

Tipo	Abreviatura
Formulario	frmFormulario
Botón	btnBoton
ComboBox	cbComboBox
Edit	txtEdit
TextEdit	txeTextEdit
Label	lblLabel

Tabla 16 Estándares de Codificación.

GLOSARIO DE TÉRMINOS

Lenguaje procedimental: En los lenguajes procedimentales el usuario indica al sistema que lleve a cabo una serie de operaciones para calcular el resultado deseado. (Facultad de Informatica, 2010)

Lenguaje orientado a objetos: Intenta simular el mundo real a través del significado de objetos que contiene características y funciones. Se clasifican como lenguajes de quinta generación. Como su mismo nombre indica, la programación orientada a objetos se basa en la idea de un objeto, que es una combinación de variables locales y procedimientos llamados métodos que juntos conforman una entidad de programación. (Lenguajes de Programacion, 2009)

Sistema Operativo: Es un programa o conjunto de programas de control que tienen por objeto facilitar el uso de la computadora y conseguir que esta se utilice eficientemente. (Conocimientos Web, 2010)

GNU/Linux: Es un Sistema Operativo. Es una implementación de libre distribución para computadoras personales, servidores y estaciones de trabajo.

Framework: Denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de software.

UML: Lenguaje Unificado de Modelado, es un lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. (Pressman, 2008)

Escalabilidad: Es la propiedad de un sistema, que indica su habilidad para manejar el crecimiento continuo de trabajo de manera fluida y para estar preparado para cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes. (Piñero, 2009)

IDE (Integrated Development Environment): Un entorno de desarrollo integrado es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un lenguaje de programación o bien, poder utilizarse para varios. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un

depurador y un constructor de interfaz gráfica. Pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. (Camacho, 2009)

Demonio o daemon (de sus siglas en inglés Disk And Execution MONitor): Es un tipo especial de proceso informático que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario (es un proceso no interactivo). Este tipo de programas se ejecutan de forma continua (infinita), vale decir, que aunque se intente cerrar o matar el proceso, este continuará en ejecución o se reiniciará automáticamente. Todo esto sin intervención de terceros y sin dependencia de consola alguna. (Watson, 2004)

Pruebas de caja negra: Se llevan a cabo sobre la interfaz de software. Se trata de demostrar que las funciones del software son operativas, que las entradas de datos se manejan de forma adecuada y que se produce el resultado esperado.