República de Cuba

Universidad de las Ciencias Informáticas

Facultad 2

Título

Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC.

Trabajo de Diploma

Presentado para optar por el título de Ingeniero en Ciencias Informáticas

Autores: Yuliani Mesa Mederos

José Alberto Betancourt Valdés

Tutor: Ing. Yordanis Tornes Medina

"Año 54 de la Revolución"

La Habana, Cuba.

Junio de 2012.

Declaración de Autoría

Por este medio declaramos que somos únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Yuliani Mesa Mederos

_____

Firma del Autor

José Alberto Betancourt Valdés

_____

Firma del Autor

Ing. Yordanis Tornes Medina

_____

Firma del Tutor

Opinión del Tutor

Un Trabajo de Diploma como el presente, del que hoy somos testigos, es la meta a que todo estudiante universitario espera llegar para obtener el anhelado título de Ingeniero en Ciencias Informáticas, para luego sentirse orgulloso de los resultados alcanzados y de poseer el conocimiento necesario para contribuir al desarrollo de la Revolución Cubana.

Durante el desarrollo del presente trabajo, los estudiantes mostraron un alto valor de independencia, originalidad, creatividad y sobre todo responsabilidad, no solo porque constituía el medio para defender su ejercicio de culminación de estudios, sino porque conocían la importancia que representa el mismo para el Proyecto en el cual se desarrolló.

Los estudiantes fueron capaces de analizar y seleccionar las herramientas idóneas para resolver el problema planteado, elementos por los cuáles el tutor considera que el resultado obtenido posee un alto valor técnico y práctico para el Proyecto 171. Como elemento representativo cabe destacar que contribuirá en gran medida a disminuir el tiempo de respuesta desde que se solicita un cambio al equipo de desarrollo del SIGESC, hasta que este cambio llegue a los entornos de despliegue. Aspecto muy valorado hoy en día por los clientes y usuarios finales de cualquier sistema informático.

Por todo lo anteriormente expresado considero que los estudiantes están aptos para ejercer como Ingenieros en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de __ puntos. Considero además que el resultado obtenido puede ser generalizado a otras aplicaciones informáticas por lo que debiera ser publicado en eventos y revistas científicas.

Ing. Yordanis Tornes Medina

_____ _____

Firma Fecha

Dedicatoria

A mis padres y mi hermano por ser lo más importante en mi vida, por quererme y darme su apoyo en todo momento. Los quiero mucho.

Dedicárselo a toda mi familia por brindarme su apoyo y estar siempre pendientes de mí.

A los amigos que siempre han estado a mi lado, por el apoyo y la confianza brindada.

Yuliani Mesa Mederos

Dedico el presente trabajo a mis padres por el apoyo constante y el sacrificio tan grande que han hecho, para sacarme adelante en todo momento. En especial a mi madre que ha sido mi principal motivación y que nunca dudó en sacrificarse para que yo lograra mis sueños. Por ser mi razón de ser, por ser mi guía en la vida, por sacrificar tus sueños por los míos. No serían suficientes las palabras para expresar las razones por las que te dedico este trabajo, que es solo el comienzo de lo que te mereces, y que estoy dispuesto a hacer para lograr que estés tan orgullosa de mí como yo lo estoy de ti. También dedico este trabajo a mi novia, a mi hermano, a toda mi familia y a mis amigos por siempre darme su apoyo.

José Alberto Betancourt Valdés

Agradecimientos

A mi mamá Caridad Mederos y mi papá Sergio Mesa por ser mi guía y darme fuerza para seguir adelante, todo lo que he logrado se lo agradezco a ellos.

A mi tía Rosalina y a mi abuelo Jesús por ser mis segundos padres, por quererme y ayudarme tanto.

A mi familia y vecinos por estar siempre pendientes de mí y darme fuerza para llegar a la meta de este largo camino, lo logré…

A mis amigas de Alquízar que aunque escogimos caminos diferentes seguimos siendo inseparables y nos hemos apoyado para lograr nuestras metas.

A mis amigas del cuarto Karina, Yamila y Yolanda, y a los chicos del 16201 por brindarme su ayuda y comprensión durante la carrera.

A mi compañero de tesis por dedicarle tanto tiempo, sacrificio y dedicación al trabajo realizado.

A todos los compañeros del proyecto 171, por ayudarnos tanto en el desarrollo de este trabajo, en especial a mi tutor Yordanis Tornes por guiarme y trabajar junto a nosotros para lograr el éxito obtenido.

A los amigos que he hecho en la Universidad durante estos cinco años, que son muy especiales, con los que he compartido momentos que nunca olvidaré.

En fin, a todas las personas con las que he compartido y me han apoyado estos 5 años.

Yuliani Mesa Mederos

Agradecimientos

Agradezco primero que todo a mis padres por apoyarme en todo momento, por ser mi guía y mi inspiración, en especial a mi mama que supo darme las fuerzas y el interés para seguir adelante en todo momento.

A mi novia por su apoyo, su ayuda y su compañía constante, por darme su amor y comprenderme siempre, sobre todo en los momentos difíciles.

A mis suegros por preocuparse por mí tanto como mis propios padres, por demostrar tanto afecto y por confiar en mí como uno más de la familia.

A nuestro tutor por su imprescindible guía, su apoyo y su abnegación durante todo el desarrollo de este trabajo.

A mi compañera de tesis por ser una compañera increíble, por su apoyo y su ayuda durante tantas horas de trabajo y sacrificio.

A todos los profesores del proyecto que de una forma u otra mostraron su apoyo incondicional y contribuyeron a la finalización de este trabajo, en especial al profesor Manfred Ramón Díaz Cabrera que con su ayuda desinteresada influyó de forma decisiva en la terminación y el resultado del presente trabajo.

A mis amigos por estar siempre dispuestos, por sus consejos y su apoyo constante y desinteresado.

José Alberto Betancourt Valdés

Resumen

El presente trabajo constituye una propuesta para solucionar los problemas existentes al generar los paquetes de instalación de las aplicaciones de escritorio del Sistema de Gestión de Emergencias de Seguridad Ciudadana (SIGESC). Recoge el resultado de haber estudiado la temática que comprende el área de generación de paquetes de instalación y actualización para aplicaciones informáticas, así como tecnologías y herramientas existentes en el mercado para automatizar esta labor.

El sistema obtenido está soportado sobre tecnología .NET para Microsoft Windows XP, específicamente sobre la versión 4.0 de este marco de trabajo. Para la implementación del mismo se utilizó el lenguaje de programación C-Sharp, el IDE de desarrollo Visual Studio 2010 y RUP como metodología de desarrollo de software para guiar el proceso. El sistema implementado posee un conjunto de funcionalidades que permitirán al equipo de desarrollo del SIGESC crear los paquetes de instalación MSI y actualización MSP requeridos sobre la tecnología de Windows Installer, mitigando la complejidad en la creación de los mismos.

Palabras claves: Actualización, Instalación, MSI, MSP, Software, Windows Installer.

Índice

Introducción

El mundo del software posee en la actualidad un papel protagónico en el desarrollo de la humanidad, dado por su vinculación a muchos de los procesos que rigen la cotidianidad. Los cambios vertiginosos en las tecnologías hacen que estas queden obsoletas en breves períodos de tiempo, generando la necesidad de nuevas soluciones que antes hubiese tomado a la humanidad docenas de años para su sustitución y hoy apenas dan cabida para actualizarse.

Ante la aparición de los sistemas informáticos surgieron nuevas interrogantes y problemas a resolver que hoy preocupan a los desarrolladores. La instalación de software es uno de estos problemas, pues desempeña un papel fundamental en el ciclo de vida de cualquier aplicación, para resolverlo es necesario utilizar mecanismos y herramientas que ayuden a mejorar el soporte técnico y mantenimiento del software en producción. El desarrollo de un software no termina con la implementación y prueba del mismo, antes de entregar el producto al usuario final, usualmente es necesario crear el instalador que guiará el proceso de instalación. En ocasiones instalar un software va más allá de una simple copia de archivos hacia el ordenador, cada día el proceso de instalación se realiza con más eficiencia, pero también adquiere mayor complejidad, por tanto no es recomendable dejar todas las acciones requeridas por la instalación en manos del usuario.

Después que un software es instalado está en constante evolución a través de extensiones como cambios en los requerimientos, cambios en la configuración o corrección de problemas. Una buena forma de aplicar estos cambios es mediante el uso de actualizaciones, estas constituyen una vía para proveer la evolución del software a los usuarios finales sin necesidad de desinstalarlo para luego ejecutar una nueva instalación, por lo que se hace necesario definir estrategias óptimas para la generación de paquetes de instalación y paquetes de actualización para los sistemas informáticos.

En la Facultad 2 de la Universidad de las Ciencias Informáticas (UCI) se implementó el Sistema de Gestión de Emergencias de Seguridad Ciudadana (SIGESC), el cual contribuye a gestionar de forma eficiente las actividades relacionadas con la atención de emergencias de seguridad ciudadana en

Venezuela. Este sistema está compuesto por una aplicación Web, un sistema de sincronización y nueve aplicaciones de escritorio.

En la actualidad los paquetes de instalación de las diferentes aplicaciones de escritorio del SIGESC se generan utilizando Visual Studio 2003. Este entorno integrado de desarrollo brinda funcionalidades para la generación de paquetes de instalación MSI (*Microsoft Installer*) pero no permite generar paquetes de actualización MSP (*Microsoft Installer Patch*), debido a ello para actualizar cualquiera de las aplicaciones de escritorio del SIGESC siempre es necesario generar un nuevo paquete de instalación en el entorno de desarrollo, para entonces desinstalar en el entorno de despliegue la versión previa e instalar el nuevo paquete generado. Todo esto demanda esfuerzo por parte del personal encargado de realizar las operaciones de actualización en la medida que aumentan las estaciones de trabajo y su dispersión geográfica es mayor. Es importante tener en cuenta además que el tamaño de un paquete de instalación es mayor que el tamaño de un paquete de actualización. Estos últimos se obtienen mediante el diferencial entre dos paquetes de instalación, por lo que su tamaño es menor, lo que puede facilitar considerablemente su distribución por la red si se decidiera implementar algún mecanismo de distribución automática de actualizaciones. Generar un paquete de instalación MSI para liberar la corrección de un problema en particular, requiere probar tanto el paquete de instalación como las funcionalidades de la aplicación en si para verificar que todos los ficheros necesarios para el correcto funcionamiento de la misma estén libres de errores, tanto los ficheros que corrigen el problema en cuestión, como el resto de ficheros no modificados pero que sin ellos la aplicación no funcionaría correctamente. Este proceso de pruebas introduce más tiempo y recursos, provocando que la actualización demore más en llegar al usuario final.

Los paquetes de instalación generados actualmente para las aplicaciones del SIGESC no chequean ni configuran, cuando es posible, los requisitos de software, configuración o hardware de las aplicaciones ya que con la herramienta utilizada se hace muy complejo incluir estas funcionalidades. Todos los requisitos deben ser chequeados o configurados manualmente antes o después de instalar cada aplicación.

Dada la situación antes expuesta surge como **problema a resolver**: ¿Cómo generar paquetes de instalación y actualización para el SIGESC permitiendo verificar y configurar los requisitos de configuración, software y hardware durante la instalación?

Teniendo como **objeto de estudio** de esta investigación el proceso de instalación y actualización de software.
El **campo de acción** se enmarca en los sistemas de generación de paquetes de instalación y actualización.

El **objetivo general** para darle solución al problema planteado consiste en: Desarrollar un sistema que permita generar los paquetes de instalación y actualización para las aplicaciones de escritorio del SIGESC de manera que chequeen durante el proceso de instalación los requisitos de configuración, software y hardware de este sistema.

Para darle cumplimiento al presente trabajo se plantean los siguientes **objetivos específicos**:

1. Especificar las características de un sistema de generación de paquetes de instalación y actualización de aplicaciones informáticas.
2. Diseñar el sistema de generación de paquetes de instalación y actualización caracterizado.
3. Implementar el sistema de generación de paquetes de instalación y actualización diseñado.

Para dar cumplimiento a los objetivos planteados se tienen como **tareas a desarrollar**:

1. Estudio de tecnologías existentes para gestionar la instalación y actualización de sistemas informáticos sobre los Sistemas Operativos.
2. Caracterización de herramientas existentes en el mercado para la generación de paquetes de instalación y actualización de sistemas informáticos.
3. Selección de la metodología y las herramientas para el desarrollo de la solución planteada.
4. Definición de las características requeridas por un sistema de generación de paquetes de instalación y actualización de aplicaciones informáticas.
5. Obtención de los modelos de diseño correspondientes a la especificación realizada.
6. Confección de los modelos de implementación del sistema a desarrollar.

7. Implementación del sistema de generación de paquetes de instalación y actualización para el SIGESC.

El presente documento consta de 3 capítulos, estructurados de la siguiente forma:

Capítulo 1: "Fundamentación Teórica"

En este capítulo se detallan los distintos conceptos relacionados con el desarrollo del sistema. Se describen diferentes tecnologías para administrar la instalación y actualización de software así como herramientas que existen para generar paquetes de instalación y de actualización. Al mismo tiempo, se analizan las diferentes herramientas, metodología de desarrollo de software, lenguaje de programación y plataforma a utilizar en la construcción de la solución propuesta.

Capítulo 2: "Presentación de la Solución Propuesta"

Describe los distintos conceptos de negocio que se utilizan en el Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC, se especifican los requisitos funcionales y no funcionales que estarán presentes en la aplicación y se definen los actores y los casos de uso del sistema.

Capítulo 3: "Construcción de la Solución Propuesta"

Se realiza el diseño del sistema a desarrollar, modelando los requisitos funcionales en diagramas de clases del diseño. Se define la arquitectura base y los patrones de diseño a utilizar. Se realiza la construcción del sistema donde los elementos del diseño se convierten en elementos de implementación en términos de componentes.

Capítulo I: Fundamentación Teórica

1.1 Introducción.

En el desarrollo de este capítulo se abordarán un grupo de aspectos y conceptos teóricos importantes que servirán de base para la investigación. Se analizan diferentes tecnologías para administrar paquetes de instalación de software y se estudian herramientas utilizadas para generar paquetes de instalación y actualización de aplicaciones informáticas. Además se exponen las principales características de la metodología de desarrollo de software, lenguaje de programación, tecnologías y herramientas que serán utilizadas para desarrollar la aplicación encargada de generar los paquetes de instalación y actualización del SIGESC.

1.2 El Software, Instalación y Actualización.

El software es un programa informático diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. En los inicios de la informática, para muchas de las aplicaciones que funcionaban en una microcomputadora bastaba con una simple copia de sus archivos para que pudieran ser utilizadas, la rápida evolución de las tecnologías, el aumento de los requerimientos de operación, entre otros aspectos, fueron aumentando la complejidad de las aplicaciones que podían ser creadas, comenzó entonces a jugar un papel fundamental un proceso de instalación que garantizara el correcto funcionamiento del software y su posterior uso.

1.2.1 Instalación del Software.

Generalmente un software está formado por el conjunto de archivos que son necesarios para su funcionamiento. Esos archivos se colocan en directorios definidos y a veces dependen de otros tipos de archivos para poder funcionar (1). Además, los programas pueden necesitar crear claves en el registro del sistema operativo (en caso de los sistemas operativos Microsoft Windows) o utilizar información que se almacena en el mismo para su correcto funcionamiento.

Estas tareas se pueden realizar manualmente, aunque en ocasiones resultan complejas porque se necesita de conocimientos avanzados sobre componentes del sistema operativo que normalmente no son

de dominio común entre los usuarios. La manipulación incorrecta de algunos de estos componentes puede traer consecuencias negativas en el funcionamiento del programa que se instala, de otros programas existentes o del propio sistema operativo, en estos casos es recomendable la utilización de un instalador: un programa que realiza las tareas requeridas para la instalación de un software sin la intervención directa del usuario en la realización de estas.

Mediante el proceso de instalación las aplicaciones desarrolladas son transferidas y configuradas apropiadamente en la computadora destino. La instalación, dependiendo del sistema desarrollado, puede consistir en copiar distintos componentes del software (bibliotecas, ejecutables, datos propios) en directorios preestablecidos del disco duro, también pueden crearse o modificarse ficheros, directorios, variables de entorno, claves y valores de registros o accesos directos.

En la actualidad es posible notar varias técnicas que permiten instalar un software en la computadora destino:

Técnica 1. La copia de los archivos necesarios en algún directorio. Este es un sistema fácil e intuitivo, uno de sus riesgos es que versiones más antiguas del mismo programa pueden quedar abandonadas en otro directorio del disco duro sin que el usuario lo note.

Técnica 2. La copia de un instalador, el que posteriormente lleva a cabo la instalación del software deseado.

Técnica 3. Existencia de algún servicio de gestión de software que se ocupe de instalar un paquete de software (sistema de gestión de paquetes).

El desarrollo de un software no se detiene cuando el sistema es entregado e instalado, sino que continúa durante el tiempo de vida del mismo. Es posible incorporar al software nuevas características que lo mejoren y hagan más estable, corrijan errores e incluyan nuevas capacidades y cualidades, este proceso es llamado actualización del software.

1.2.2 Actualización del Software.

Históricamente, el desarrollo de las aplicaciones informáticas se comportaba de forma estática. Se obtenía el software, se instalaba y se utilizaba tal como era distribuido originalmente hasta que se liberaba una próxima versión. Hoy en día este modelo no se aplica de igual forma, el mundo digital actual está en constante cambio y con el fin de que los últimos avances estén disponibles de inmediato, el software es ahora mucho más dinámico (2).

Debido a la complejidad de los procesos a informatizar, comúnmente los sistemas informáticos son puestos en producción con errores no detectados que luego son reportados por los usuarios finales. Una de las vías para corregir estos errores lo antes posible e introducir mejoras a un software es a través de las actualizaciones. El proceso de actualización puede ser visto como el movimiento de una configuración a otra por adición, eliminación, reemplazo o reconfiguración de las funcionalidades del software (3).

Una actualización contiene alteraciones relacionadas con las funcionalidades y la configuración de un sistema informático. El uso de actualizaciones representa grandes ventajas para los desarrolladores, esta práctica no solo permite corregir los posibles errores en un menor tiempo, también da la posibilidad de cambiar solo los archivos estrictamente necesarios para resolver el problema existente. Generalmente las actualizaciones están contenidas en archivos pequeños, haciendo más rápido su tráfico por la red y reduciendo el tiempo de instalación de la misma ya que realizan menos operaciones que un paquete de instalación.

Las actualizaciones no deben utilizarse para cambios de gran envergadura y sus efectos están limitados, no pueden eliminar componentes o características, cambiar los códigos de identificación del producto, ni eliminar o cambiar los nombres de los archivos o las claves del registro (4).

1.3 Tecnologías para Administrar la Instalación del Software.

Los sistemas operativos disponen de tecnologías para administrar la instalación y actualización de las aplicaciones informáticas. En el presente epígrafe se analizarán mecanismos utilizados con este objetivo en diferentes sistemas operativos.

1.3.1 Gestor de Paquetes RPM.

El gestor de paquetes RPM[1] es un potente sistema de gestión de paquetes capaz de instalar, desinstalar, verificar, consultar y actualizar programas informáticos. Cada paquete de software contiene el conjunto de ficheros a instalar junto con la información sobre el propio paquete como nombre, versión y descripción. Además contiene una librería API[2] que permite a los desarrolladores gestionar dichas operaciones a partir de lenguajes de programación como C o Python. RPM es software libre, publicado bajo la licencia GNU/GPL[3] y es utilizado en muchas distribuciones de Linux (5).

La historia de RPM está definitivamente ligada a GNU/Linux, especialmente a la distribución *Red Hat Enterprise Linux.* Con el tiempo, esta distribución se ha convertido en una de las distribuciones más populares disponibles hoy en día, y en gran medida RPM es el responsable (6).

La facilidad de uso de RPM radica en su capacidad de manejar automáticamente los archivos de configuración de los paquetes que se instalan. RPM está incluido en diferentes distribuciones Linux además de *Red Hat*, entre las que se destacan: *Fedora Linux*, *Mandriva Linux* y *SuSE Linux*.

1.3.2 Gestor de Paquetes DPKG.

DPKG fue originalmente creado por *Matt Welsh*, *Carl Streeter* e *Ian Murduck* y escrito en ese entonces en el lenguaje de programación Perl. Posteriormente el programa fue reescrito en C por *Ian Jackson* en 1993. DPKG es una abreviatura de *Debian package* (paquete de Debian). Se trata de un equipamiento lógico utilizado para la gestión de paquetes en *Debian Linux* y otras distribuciones de GNU/Linux como es el caso de *Ubuntu Linux*. Es una herramienta de bajo nivel que permite la instalación, desinstalación y consulta de información de paquetes con extensión *.deb*. Existen herramientas de alto nivel como *APT*, *aptitude* o S*ynaptic que* lo utilizan (7) y hacen más fácil la gestión para los usuarios finales.

[1] RPM: Del ingles Red Hat Package Manager.

[2] API: Del inglés *Application Programming Interface*.Interfaz de programación de aplicaciones en español. Es un conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación.

[3] GNU/GPL: La Licencia Pública General de GNU o más conocida por su nombre en inglés GNU *General Public License.*

La herramienta APT surge debido a que los desarrolladores del gestor de paquetes DPKG necesitaban un modo fácil, rápido y eficiente de instalar los programas, que manejara automáticamente las dependencias y se hicieran cargo de la configuración mientras estos programas se actualizaban. Este programa ha sido adaptado para usarse con RPM y ha sido adoptado por otras distribuciones de Linux además de *Debian* (8). Existen programas que proporcionan una interfaz gráfica para APT como *Synaptic*.

1.3.3 Setup API.

Setup API ofrece un conjunto de funciones que los instaladores pueden utilizar para realizar operaciones de instalación en los sistemas operativos *Microsoft Windows*. Los programas que utilizan esta tecnología para llevar a cabo el proceso de instalación son aplicaciones con extensión *.exe* (abreviación del inglés *executable*). Estos ficheros pueden descomprimirse para abrir una serie de documentos, iniciar una aplicación o sencillamente pueden instalar un programa.

Setup API ya no es comúnmente utilizado para la instalación de aplicaciones informáticas sobre *Microsoft Windows*, en su lugar se propone utilizar *Windows Installer,* tecnología recomendada por *Microsoft* para administrar los paquetes de instalación y actualización (9)**.**

1.3.4 Windows Installer.

Microsoft Windows Installer es un servicio de instalación y configuración que se distribuye como parte de la familia de sistemas operativos Microsoft Windows (a partir de la versión *Windows 2000*) (10).

Con *Windows Installer* todos los equipos mantienen una base de datos de información sobre cada aplicación que se instala, incluidos los componentes y las claves del registro. Cuando se desinstala una aplicación, se comprueba la base de datos para asegurarse de que ninguna otra aplicación precisa un archivo, clave de registro o componente antes de quitarlo, lo que evita que la eliminación de una aplicación afecte al buen funcionamiento de otra (10).

Windows Installer también provee reparación automática, es decir, brinda la posibilidad de que una aplicación reinstale automáticamente los archivos que falten en caso de que el usuario los haya eliminado accidentalmente. Proporciona además la posibilidad de revertir una instalación (*rollback*).

La tecnología *Windows Installer* ofrece:

- Formato del paquete de *Windows Installer*: Constituye una base de datos relacional con toda la información necesaria para describir la forma en que se instala una aplicación.
- Instalación personalizada: Permite que los paquetes de instalación sean personalizados, logrando así un resultado más cercano a las necesidades, no solo del desarrollador sino también del usuario final, ya que los cambios van desde la interfaz hasta operaciones relacionadas directamente con la instalación de la aplicación.
- Transacción de servicios de instalación: Servicio que ayuda a garantizar que el sistema operativo permanezca en un estado coherente, incluso en el caso de una instalación fallida o cancelada.

Los paquetes de *Windows Installer* se distribuyen a través de archivos con extensión .msi, en lo adelante MSI (*Microsoft Installer* o Paquete de *Windows Installer*) y archivos con extensión .msp, en lo adelante MSP (*Microsoft Installer Patch* o Revisión de *Windows Installer*).

1.3.4.1 MSI.

Los paquetes MSI se pueden definir como paquetes de software que contienen la información necesaria para automatizar determinadas operaciones o acciones que requiere una aplicación para instalarse sin la intervención directa del usuario. Estos contienen una base de datos estructurada que almacena todas las instrucciones y los datos requeridos para instalar y desinstalar las aplicaciones.

Un paquete está compuesto por dos elementos principales: componentes y rutas maestras.

Componentes:

Un componente es la mínima parte de un paquete de instalación. Cada componente es tratado por *Windows Installer* como una unidad. Los componentes pueden contener archivos, grupos de archivos,

directorios, componentes COM[4], claves del registro de *Windows*, accesos directos y datos de otro tipo. Cada componente está identificado globalmente por un GUID[5].

Rutas maestras *(Key paths):*

Una ruta maestra es un fichero específico, clave de registro, o fuente de datos ODBC[6] que el autor del paquete especifica como crítico para un componente dado. Como las rutas maestras más utilizadas son en forma de fichero, se suele utilizar el término fichero maestro (*keyfile*). Un componente puede contener, a lo sumo, una ruta maestra; si un componente no tiene establecida de manera explícita una ruta maestra, el directorio destino del componente es tomado como la ruta maestra. Cuando se ejecuta un instalador basado en MSI, *Windows Installer* comprueba la existencia de estos ficheros críticos o claves de registro. Si existe un desajuste entre el estado actual del sistema y el valor especificado en el paquete MSI, entonces la característica asociada es reinstalada. Este proceso es también conocido como auto-reparación.

Los paquetes MSI llevan a cabo todas las tareas propias de la instalación, tales como copiar archivos a un disco duro, realizar modificaciones en el registro, crear variables de entorno y crear accesos directos.

1.3.4.2 MSP.

Con frecuencia, luego que una aplicación determinada es instalada, es posible que la misma necesite ser actualizada con el objetivo de realizar cambios para solucionar problemas, o simplemente para optimizar el código fuente para mejorar el desempeño de la misma. Es entonces cuando se hace uso de los archivos MSP para aplicaciones instaladas mediante paquetes MSI. Estos archivos son llamados Paquetes de Revisión de *Windows Installer* o *Parche de Microsoft Windows Installer*.

[4] COM: Component Object Model. Plataforma de Microsoft para componentes de software utilizada para permitir la comunicación entre procesos.

[5] GUID: Globally Unique Identifier. Un Identificador Globalmente Único, es un número aleatorio empleado en aplicaciones de software.

[6] ODBC: *Open Data Base Connectivity*, es un estándar de acceso a bases de datos desarrollado por *SQL Access Group* en 1992.

Similar a los paquetes MSI, los paquetes MSP contienen una base de datos con toda la información necesaria para llevar a cabo el proceso de actualización. La creación de un paquete MSP se realiza comparando dos paquetes MSI de diferentes versiones con el objetivo de encontrar las diferencias entre ambos, a partir de esta información se genera un paquete que solo contenga los archivos necesarios para actualizar la aplicación instalada.

Los paquetes de actualización representan una gran ventaja para el usuario final, permiten corregir errores o mejorar el funcionamiento de cualquier aplicación con solo ejecutar el parche indicado en la computadora donde se encuentra instalada la aplicación a actualizar, sin necesidad de desinstalarla previamente. Además, ayudan a que los cambios lleguen al usuario lo antes posible y de una manera más fácil, ya que el tamaño de un paquete MSP es menor que el tamaño de un paquete MSI.

1.4 Herramientas para Generar Paquetes de Instalación y Actualización.

Existen diferentes herramientas que explotan las tecnologías estudiadas para la gestión de paquetes de instalación y actualización de aplicaciones informáticas en los sistemas operativos *Microsoft Windows*, con el objetivo de encontrar una solución que pueda ser utilizada para crear los paquetes de instalación y actualización del SIGESC, aplicación desarrollada para la familia de sistemas operativos antes mencionada, específicamente *Windows XP,* a continuación se ofrece un resumen de las características más significativas de las herramientas estudiadas.

1.4.1 Inno Setup.

Inno Setup es una herramienta para generar scripts e instaladores de programas, es de código abierto y posee compatibilidad con casi todas las versiones de *Microsoft Windows*, especifica paso a paso los datos que son necesarios para configurar el instalador de la aplicación a instalar**.** Permite configurar y personalizar el proceso de instalación de los archivos que se copiarán en el ordenador, de modo tal que se pueden seleccionar los componentes, crear entradas de registro y crear accesos directos en diferentes ubicaciones del sistema operativo. Brinda además la posibilidad de especificar el directorio donde se van a copiar los archivos de la instalación.

Si al ejecutar un instalador creado con *Inno Setup*, este detecta que el programa en cuestión ya está instalado, compara los elementos que los componen y si la versión a instalar es diferente y superior, instala los elementos necesarios para actualizar el programa y a la vez efectúa el registro de los mismos.

Algunas de las principales características de ***Inno Setup*** son (11)**:**

- Creación de instaladores de extensión *.exe*, con soporte multilenguaje, para facilitar la distribución y la instalación de las aplicaciones.
- Soporte para la generación de instaladores encriptados con contraseña.
- Posibilidad de configurar el tipo de instalación que debe realizar el instalador que se está generando: completa, mínima o personalizada.
- Capacidad para añadir iconos y accesos directos en el menú de Inicio de *Windows*, una vez concluida la instalación.

Actualmente *Inno Setup* no planea desarrollar una versión basada en la tecnología *Windows Installer,* por lo que no dispone de funcionalidades para crear paquetes de Instalación MSI (12). *Inno Setup* no permite crear paquetes de actualización, este proceso se lleva a cabo creando un instalador con los nuevos archivos y sobrescribiendo el paquete instalado.

1.4.2 BitRock InstallBuilder.

BitRock InstallBuilder es una herramienta de desarrollo para la construcción de instaladores para aplicaciones de escritorio y software de servidores, fue creada en el 2003 en Sevilla, España por Daniel López Ridruejo.

Crea fácilmente instaladores multiplataforma para los sistemas operativos *Linux, Windows, Mac OS X, FreeBSD, OpenBSD y Solaris*. La aplicación genera instaladores nativos que tienen una apariencia equivalente a la del sistema operativo en el que se ejecutan, es decir que se integra en su entorno gráfico sin necesidad de dependencias. Esta herramienta genera ficheros de instalación ejecutables optimizados

en tamaño y velocidad de instalación, independientemente de factores externos, que pueden ejecutarse con interfaz de usuario, en modo texto o silencioso (13).

Posee una licencia libre totalmente funcional para los proyectos de código abierto. De forma predeterminada, InstallBuilder realiza una copia de seguridad de todos los archivos sobrescritos durante la instalación para que puedan ser recuperados en caso de un error. El sistema muestra los componentes y las características disponibles en una estructura de árbol, permitiendo al usuario seleccionar solo aquellos que desea instalar.

BitRock InstallBuilder se distingue por (13):

- Ser independiente de plataformas o versiones
- Tener un solo proyecto XML[7] para todas las instalaciones.
- Generar instaladores para aplicaciones desarrolladas en diferentes lenguajes de programación como por ejemplo: Java, PHP, Perl, Python, C# o C++.
- Proporcionar una herramienta de construcción de línea de comando.

BitRock InstallBuilder al igual que la herramienta analizada anteriormente, tampoco genera paquetes de instalación para *Windows Installer* y no posee una forma cómoda de establecer los requisitos de instalación, estas acciones se tienen que realizar mediante configuración de scripts que deben ser adicionados al proyecto para ser ejecutados, o modificando el archivo XML que contiene la estructura del proyecto.

1.4.3 Microsoft Visual Studio.

La herramienta *Microsoft Visual Studio* es un Entorno de Desarrollo Integrado que contiene dentro de sus funcionalidades la creación de proyectos de instalación. El paquete de instalación para *Windows Installer* construido a partir de un proyecto creado con la herramienta *Microsoft Visual Studio Installer*, contiene todos los datos e instrucciones necesarias para instalar una aplicación.

[7] XML: Lenguaje de Marcas Extensible (en inglés: eXtensible Markup Language).

Microsoft Visual Studio Installer permite (14):

- Construir proyectos de Instalación que contengan las salidas de más de un producto desarrollado en *Microsoft Visual Studio*. El proyecto de instalación puede incluir archivos de salida de *Visual Basic*, *Microsoft Visual C++* y *Microsoft Visual J++.*
- Crear y configurar componentes para instalarse en la máquina destino.
- Especificar dónde colocar los archivos y las carpetas necesarias para la instalación.
- Manipular el registro de la máquina destino.
- Crear accesos directos en lugares definidos por el usuario.
- Distribuir el paquete de instalación (archivo *.msi*) con toda la información necesaria para la instalación con archivos cabinet (*.cab*) comprimidos o descomprimidos, dependiendo de sus requerimientos y capacidades de distribución. El archivo .cab es el formato nativo de archivo comprimido de Microsoft Windows.

Microsoft Visual Studio Installer permite generar varios tipos de proyectos de instalación: proyecto de módulo de combinación *.msm*, proyecto de instalación *.msi*, proyecto de instalación Web *.msi*, proyecto CAB *.cab* y asistente para la instalación *.msi, pero* no incluye un proyecto de instalación relacionado con la creación de paquetes de *Microsoft Installer Patch* (archivo *.msp*) para que los desarrolladores puedan actualizar sus aplicaciones*,* por ende cuando se necesita actualizar una aplicación, siempre es necesario crear un nuevo paquete de instalación para realizar este proceso aunque la diferencia de un paquete a otro sea mínima. Esta herramienta permite crear paquetes de instalación con varias prestaciones, pero no posee interfaces que le permitan al usuario realizarlas de forma sencilla y rápida sin necesidad de adentrarse en el código de la herramienta. Para esto se tienen que programar acciones personalizadas para ser incluidas en el proyecto y sean ejecutas en el proceso de instalación.

1.4.4 InstallShield.

Es una herramienta destinada a crear paquetes de instalación de *software*. *InstallShield* permite crear instaladores para aplicaciones de escritorio, web, servidores y móviles. Es capaz de generar varios tipos de instaladores, entre los que se incluyen paquetes de instalación MSI y paquetes de actualización MSP. Incorpora funcionalidades que permiten crear un tipo de proyecto llamado *InstallShield Suite*, con el objetivo de agrupar múltiples productos (Instaladores) en un mismo paquete de instalación (15).

InstallShield posee características como (15):

- Asistente de Proyectos: Este asistente guía de una manera intuitiva a través del proceso de creación de instalaciones, reduciendo de manera significativa el tiempo.
- Entorno de diseño intuitivo: Su entorno de diseño avanzado hace la creación de instalaciones más eficiente indicando claramente los pasos fundamentales relacionados con la creación de una instalación. También ofrece vistas separadas para las tareas comunes de cada paso.
- Herramientas de automatización: Puede automatizar sus procesos de generación creándolos desde la línea de comandos con sus potentes herramientas.

Posee funcionalidades tales como permitirle al usuario la configuración de algunos requisitos de instalación predefinidos, por ejemplo la comprobación del sistema operativo instalado en la máquina destino o la verificación de que una aplicación determinada esté instalada. Además permite que el usuario establezca un requisito personalizado basado en la búsqueda de un archivo, carpeta, registro o componente en la computadora. Permite crear accesos directos de las aplicaciones a instalar, así como la estructura de directorios requerida.

Esta herramienta, a pesar de poseer un grupo de funcionalidades que pudieran resolver los problemas existentes actualmente en la creación de los paquetes de instalación y actualización del SIGESC, es una herramienta con licencia propietaria y su utilización requiere de una inversión monetaria que no se ha podido asumir, por lo que no ha podido ser utilizada por el equipo de desarrollo del SIGESC desde que comenzó la puesta en producción del mencionado sistema.

1.4.5 Windows Installer XML.

Windows Installer XML (WiX) es una herramienta liberada bajo los términos de la licencia CPL[8], desarrollada originalmente por Rob Mensching para *Microsoft*. WiX permite construir paquetes de instalación MSI y paquetes de actualización MSP para la tecnología *Windows Installer* a partir de uno o más archivos de configuración en formato XML, donde se especifica la estructura y configuración del futuro paquete de instalación o actualización, esto permite crear un instalador totalmente adaptado a las necesidades de los desarrolladores. Posee un conjunto de herramientas desarrolladas en CSharp (C#) y requiere de .NET Framework para funcionar. Entre estas herramientas se encuentra un compilador y un enlazador que se encargan de compilar y generar el paquete de instalación o actualización a partir de los archivos de configuración XML. El compilador de WiX es el encargado de comprobar que la estructura XML del archivo de entrada con extensión *.wxs* es la correcta, genera un nuevo archivo de extensión *.wixobj* a partir del cual el enlazador construirá el paquete de instalación. Para construir el paquete de actualización, el enlazador de WiX a partir del archivo *.wixobj* genera un archivo *.pcp* el cual será utilizado por la herramienta Msimsp.exe para generar el paquete de actualización MSP. Por lo tanto, WiX pudiera ser parte de cualquier aplicación de generación de paquetes de instalación y actualización de alta o baja complejidad.

De todas las versiones disponibles la versión 2.0 es la más antigua, esta fue liberada el 5 de octubre de 2007. Por ser una de las primeras liberaciones es considerada la versión con más errores conocidos, la mayoría de estos errores fueron corregidos mediante actualizaciones. A pesar de estos errores WiX 2.0 fue un acontecimiento significativo, especialmente para los desarrolladores interesados en generar sus propios instaladores.

Posterior a esta versión está la versión 3.0 de WiX que fue liberada el 19 junio de 2009, constituyó desde sus inicios un avance revelador sobre su predecesora. El desarrollo de la misma tardó aproximadamente cuatro años y medio, incluyendo suficientes mejoras que la convirtieron en la primera liberación verdaderamente estable de este proyecto. WiX 3.0 fue diseñado para integrarse con Visual Studio 2005 y Visual Studio 2008, esto incorpora un nuevo tipo de proyecto a la herramienta *Microsoft Visual Studio,* llamado *Windows Installer XML*, que permite la creación de proyectos relacionados con la creación de

[8] CPL**:** Licencia Pública Común (en inglés:*Common Public License*).

instaladores usando el conjunto de herramienta WiX. La última versión estable de WiX es la 3.5 y se puede integrar oficialmente con *Visual Studio* 2005, 2008 y 2010 (15). La interfaz de usuario incorporada en los instaladores creados con estas herramientas fue considerablemente mejorada con respecto a las versiones anteriores.

1.5 Conclusiones Parciales.

Las herramientas de generación de paquetes de instalación y actualización descritas con anterioridad, no cumplen en su totalidad con los requisitos necesarios para solucionar los problemas existentes en la creación de los paquetes de instalación y actualización de las aplicaciones de escritorio del SIGESC. Como se describió en epígrafes anteriores algunas de estas herramientas no generan paquetes de actualización, no permiten incluir el chequeo de requisitos o bien el mecanismo de que disponen es engorroso, otras no están diseñadas para *Windows Installer*, tecnología utilizada actualmente para administrar la instalación de las aplicaciones de escritorio del SIGESC, o bien, como es el caso de InstallShield que pudiera resolver los problemas actuales, no puede ser utilizada por el tipo de licencia que posee. El conjunto de herramientas WiX ayuda a darle solución a la problemática existente, sin embargo, WiX no posee una interfaz gráfica que facilite a los usuarios la creación de paquetes de instalación y actualización, todas las acciones necesarias requieren de la configuración de un archivo XML que posteriormente debe ser compilado utilizando una herramienta de línea de comandos para luego generar el paquete de instalación o actualización utilizando otra herramienta de línea de comando. Por lo antes expuesto se decide desarrollar una aplicación, que haciendo uso de las herramientas WiX, permita generar de una forma más amigable los paquetes de instalación y actualización de las aplicaciones de escritorio del SIGESC.

1.6 Metodología, Tecnologías y Herramientas de Desarrollo de Software.

Para el desarrollo de la solución se utilizaron un conjunto de herramientas y tecnologías, de las cuales varias forman parte del sistema de gestión de la configuración del SIGESC.

1.6.1 Metodología de Desarrollo de Software.

Una metodología de desarrollo es una colección de documentación formal referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software. Son el conjunto de métodos, técnicas y herramientas utilizadas para alcanzar un objetivo. Definir la metodología correcta es de vital importancia pues es la encargada de guiar todo el proceso de desarrollo del software, es la que define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

Existen dos clasificaciones atendiendo a la filosofía de desarrollo del software, las metodologías ágiles y las tradicionales. Las ágiles enfatizan las comunicaciones cara a cara con el cliente en vez de llevar una documentación rígida. El cliente está en todo momento colaborando en el proyecto y es más importante la capacidad de respuesta ante un cambio. Las tradicionales se centran en el control del proceso, implantando los artefactos que se deben originar, las actividades implicadas, las herramientas y notaciones que se usarán.

1.6.1.1 Proceso Unificado de Software.

El Proceso Unificado de Rational, RUP por sus siglas en inglés, es un proceso de ingeniería de software planteado por Kruchten en 1996, cuyo objetivo es producir software de alta calidad, es decir, que cumpla con los requerimientos de los usuarios dentro de una planificación y presupuesto establecidos. Se centra en la definición detallada de los procesos, tareas y herramientas a utilizar, y propone una extensa documentación. Cubre todo el ciclo de vida en el desarrollo del software (16).

Algunas de las razones para seleccionar RUP como metodología de desarrollo son:

- RUP posee varios elementos de planificación (Plan de Desarrollo, Plan de Iteración, entre otros) que permiten llevar el control del desarrollo del proyecto.
- La UCI está en constante retroalimentación de estudiantes, quienes constituyen una parte importante del equipo de desarrollo. La documentación detallada generada con RUP, sirve de base para futuros desarrolladores.

- RUP fue la metodología seleccionada por el equipo de desarrollo del SIGESC para guiar y mantener el desarrollo de este sistema.

- Al integrar RUP con el Lenguaje Unificado de Modelado (UML, siglas del inglés Unified Modeling Language) constituye una de las metodologías estándar más utilizada para el análisis, documentación e implementación de sistemas orientados a objetos.

1.6.2 Lenguaje de Modelado.

El Lenguaje Unificado de Modelado es un lenguaje gráfico que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema de software. Facilita la compresión, diseño, configuración y mantenimiento de los sistemas que se deben construir (17).

Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que ha sido diseñado para modelar cualquier tipo de proyecto. UML permite crear varios tipos de diagramas, entre ellos los diagramas de casos de uso, diagramas de clases, de componentes y despliegue. Se puede emplear en el desarrollo de un software como soporte a una metodología de desarrollo como RUP, pero no se especifica que metodología o proceso se debe usar.

1.6.3 Herramienta CASE[9].

Las herramientas CASE se pueden definir como un conjunto de programas que asisten a los desarrolladores, durante todas las etapas en el ciclo de desarrollo de un software. Estas herramientas son un elemento primordial en el diseño y la construcción del software, permiten no solo graficar los procesos definidos, sino que simplifican y agilizan el trabajo de los programadores (18).

1.6.3.1 Visual Paradigm para UML.

Visual Paradigm es una herramienta CASE creada para contemplar el ciclo completo del desarrollo del software. Permite modelar diagramas que faciliten la construcción del sistema. Proporciona características tales como generación de código, ingeniería inversa y generación de informes. Incorpora el soporte para

[9] CASE: Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora.

trabajo en equipos, permitiendo que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios realizados.

Sus principales características son:

- Brinda numerosos estereotipos a utilizar, lo cual permite una mejor compresión de los diagramas.
- Permite redactar especificaciones de casos de uso sin necesidad de utilizar un editor de texto externo.
- Posibilita generar código fuente a partir de los diagramas diseñados para plataformas como .NET y Java, así como obtener diagramas a partir del código fuente existente.
- Se integra fácilmente con ambientes de desarrollo integrados como Visual Studio.
- Se puede integrar con el sistema de control de versiones Subversion facilitando el trabajo colaborativo.

1.6.4 Plataforma de Desarrollo.

Una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación y posterior ejecución de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo, también es posible encontrarla ligada a una familia de lenguajes de programación o a una Interfaz de Programación de Aplicaciones (API por sus siglas en inglés).

1.6.4.1 .Net Framework 4.0.

Microsoft .NET Framework es un modelo de programación integral y coherente de *Microsoft* para crear aplicaciones que proporcionen magníficas experiencias de usuarios visuales, así como una comunicación eficaz y segura, y la capacidad de modelar una amplia gama de procesos empresariales (19).

Microsoft .NET Framework 4.0 es una plataforma de software que integra disímiles tecnologías facilitando el trabajo entre diferentes lenguajes de programación y librerías con el objetivo de concebir aplicaciones e integrarlas a otros sistemas previamente creados (20).

La herramienta WiX seleccionada fue desarrollada sobre .NET, con el objetivo de seguir la línea de desarrollo del SIGESC se decidió implementar la solución sobre esta misma plataforma. Para aprovechar las mejoras que ofrece el Framework 4.0 se seleccionó esta versión del Framework para la implementación de la solución.

1.6.5 Entorno de Desarrollo Integrado.

Un Entorno de Desarrollo Integrado (IDE, siglas del inglés *Integrated Development Environment*) es un programa informático compuesto por un conjunto de herramientas de programación. Puede consistir en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI por sus siglas en inglés). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Es posible que un mismo IDE pueda soportar varios lenguajes de programación, tal es el caso de Visual Studio.

1.6.5.1 Microsoft Visual Studio 2010.

Microsoft Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones para móviles. *Visual Basic*, *Visual C#* y *Visual C++* utilizan todos el mismo entorno de desarrollo integrado, que habilita el uso compartido de herramientas y facilita la creación de soluciones en varios lenguajes. Así mismo dichos lenguajes utilizan las funciones de *.NET Framework* que ofrecen acceso a tecnologías claves para simplificar el desarrollo de las aplicaciones (21).

Visual Studio 2010 incluye numerosas mejoras, entre la que se destacan (22)**:**

- **Mejoras visuales:** Se ha rediseñado el IDE con el fin de mejorar la legibilidad. Se han quitado las líneas y los degradados innecesarios para conseguir una mayor claridad.

- **Editor de código:** El nuevo editor de código facilita la lectura del código. Puede acercar el texto si presiona CTRL y mueve la rueda del mouse. Además, al hacer clic en un símbolo en Visual C# todas las instancias de ese símbolo se resaltan automáticamente.

Microsoft Visual Studio 2010 posee una serie de características nuevas que fortalecen no solo el trabajo en el entorno, sino también el aprendizaje y entrenamiento de los desarrolladores.

1.6.6 Lenguaje de Programación C# 4.0.

C# o *CSharp es* un lenguaje de programación orientado a objetos derivado de la evolución de los lenguajes C y C++ junto a las mejores características de lenguajes como Java y Visual Basic. Está formado por un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

C# es simple, eficaz y con seguridad de tipos. Con sus diversas mejoras permite desarrollar aplicaciones rápidamente, al mismo tiempo que mantiene la expresividad y elegancia de los lenguajes de tipo C. Presenta considerables mejoras e innovaciones en áreas como control de versiones, eventos y recolección de elementos no utilizados (liberación de memoria). A diferencia de otros lenguajes admite herencia simple de clases, presenta un modificador llamado *internal* y la estructura iterativa *foreach* (23).

Con la aparición de Visual Studio 2010 surge la versión 4.0 de C# la cual se decidió utilizar para implementar la solución. La selección de este lenguaje se justifica principalmente en que C# es un lenguaje que explota a fondo la plataforma .NET pues fue diseñado específicamente para ella, careciendo de elementos heredados innecesariamente.

1.7 Conclusiones.

El estudio realizado en el capítulo permitió afianzar los conocimientos necesarios entorno a la temática que comprende la generación de paquetes de instalación y actualización para la tecnología *Windows Installer* presente en el sistema operativo *Windows XP*, necesarios para comprender en su totalidad el problema planteado en este trabajo.

La elección del conjunto de herramientas WiX posibilitará diseñar una solución que satisfaga las necesidades actuales del equipo de desarrollo del SIGESC para la creación de los paquetes de instalación y actualización de las aplicaciones de escritorio del mencionado sistema.

La metodología, tecnologías y herramientas seleccionadas para la construcción de la solución objeto del presente trabajo, contribuirán de forma positiva a la ejecución del proceso de desarrollo facilitando la continuidad del mismo una vez concluida esta etapa de trabajo.

Capítulo II: Presentación de la Solución Propuesta

2.1 Introducción.

Este capítulo describe la propuesta de solución del Sistema de Generación de Paquetes de Instalación y Actualización para las aplicaciones del SIGESC. La modelación relativa al conocimiento que se expone proporciona un nivel de descripción del negocio que permitirá mediante un diagrama de clases, mostrar al usuario los principales conceptos que se utilizan en el sistema a desarrollar. Esto ayuda a los usuarios, desarrolladores e interesados, a utilizar un vocabulario común para poder entender el contexto en que se emplaza el sistema. Para identificar correctamente los requisitos y poder construir el sistema que se propone, se necesita haber realizado un análisis previo del objeto de estudio. Estos requisitos estarán presentes durante todo el desarrollo del sistema dejando una traza en cada etapa del mismo.

2.2 Descripción del Sistema.

El Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC consistirá en una aplicación de escritorio para el sistema operativo *Microsoft Windows XP*, y surge con el objetivo de resolver un grupo de problemas que existen a la hora de generar los paquetes de instalación y actualización de las aplicaciones de escritorio del SIGESC. El desarrollo de la propuesta de solución comprende dos escenarios fundamentales: la creación de paquetes de instalación MSI y la creación de paquetes de actualización MSP sobre la tecnología *Windows Installer.*

Para la creación de un proyecto de instalación MSI el sistema permitirá registrar la información referente al paquete de instalación que se está creando, por ejemplo: nombre del producto, identificador del producto, código de actualización, versión del producto y fabricante. Debe permitir agregar componentes como archivos dll, ejecutables, imágenes, así como todos aquellos archivos necesarios para el correcto funcionamiento de la aplicación a instalar, además de crear la estructura de directorios donde estarán ubicados estos archivos.

El sistema brindará la posibilidad de que el usuario establezca determinados requisitos que podrán ser comprobados o creados durante el proceso de instalación para que la aplicación al instalarse funcione

correctamente. Estos requisitos pueden ser de hardware, software o de configuración de ciertos elementos como variables de entorno o claves del registro del sistema operativo *Microsoft Windows XP.*

Otra de las funcionalidades del sistema consistirá en crear los accesos directos que tendrán las aplicaciones al instalarse. Además de crear proyectos para generar paquetes de instalación y actualización, el sistema también permitirá guardar los proyectos creados con el objetivo de no perder el trabajo realizado y posibilitará además abrir proyectos existentes.

Para la creación de un proyecto de actualización MSP**,** el sistema creará un paquete de actualización a partir de dos paquetes de instalación MSI pertenecientes a una misma aplicación pero con diferentes versiones, versión actual y versión superior. Permitirá registrar también información referente al paquete de actualización como por ejemplo: nombre de la actualización, clasificación de la actualización y la descripción de la actualización.

Todas las configuraciones para crear los paquetes de instalación MSI y los paquetes de actualización MSP se guardarán en un archivo en notación XML, este archivo será compilado utilizando el compilador (candle.exe) de WiX para comprobar que la estructura XML del archivo de entrada con extensión .*wxs* es la correcta, obteniendo como resultado un nuevo archivo de extensión .*wixobj* a partir del cual el enlazador (light.exe) de WiX generará los paquetes con extensión *.msi y .msp*.

2.3 Modelo de Dominio.

Un Modelo de Dominio es una representación visual de clases conceptuales o de objetos reales en un dominio de interés. Representa las clases conceptuales del dominio del problema, lo cual puede ser una idea o un objeto físico (símbolo, definición y extensión) (24).

El Modelo de Dominio se representa con UML, a través de un Diagrama de Clases donde se muestran (24):

- Conceptos u objetos del dominio del problema: clases conceptuales.
- Asociaciones entre clases conceptuales.
- Atributos de las clases conceptuales.

Teniendo en cuenta que en el proceso estudiado no están definidos claramente los límites de las actividades que se desarrollan, se decide realizar un modelo de dominio para representar los principales conceptos e ideas propias del dominio donde se enmarca la solución propuesta al problema a resolver.

2.3.1 Clases Conceptuales.

Paquete: Representa el paquete de instalación o actualización que se desea crear.

Paquete MSI: Representa el paquete de instalación MSI.

Paquete MSP: Representa el paquete de actualización MSP.

Producto: Representa la aplicación a la que se le crea un paquete de instalación o un paquete de actualización.

Componente: Es la unidad más pequeña que se va a instalar, puede constar de muchos elementos diferentes, incluidos archivos, accesos directos y claves de Registro. Representa todos los componentes por los que está compuesto un paquete MSI o un paquete MSP.

Archivo: Es una referencia a un archivo individual que instala o actualiza un paquete MSI o un paquete MSP respectivamente, dígase dll, ejecutables, imágenes etc. Los archivos están incluidos en un elemento Componente.

Directorio: Representa la estructura de directorio donde van a estar ubicados todos los componentes.

Acceso Directo: Representa los accesos directo asociados a un archivo.

Característica: Una característica agrupa un conjunto de componentes. Las Características son las que el usuario definirá para la instalación.

Media: Es donde se empaquetan todos los archivos que se van a instalar o a actualizar.

Condición: Representa los requisitos o condiciones necesarias para el correcto funcionamiento del Producto a instalar, estas pueden ser chequeadas o configuradas por el paquete MSI durante la ejecución de la instalación.

2.3.2 Diagrama del Modelo de Dominio.

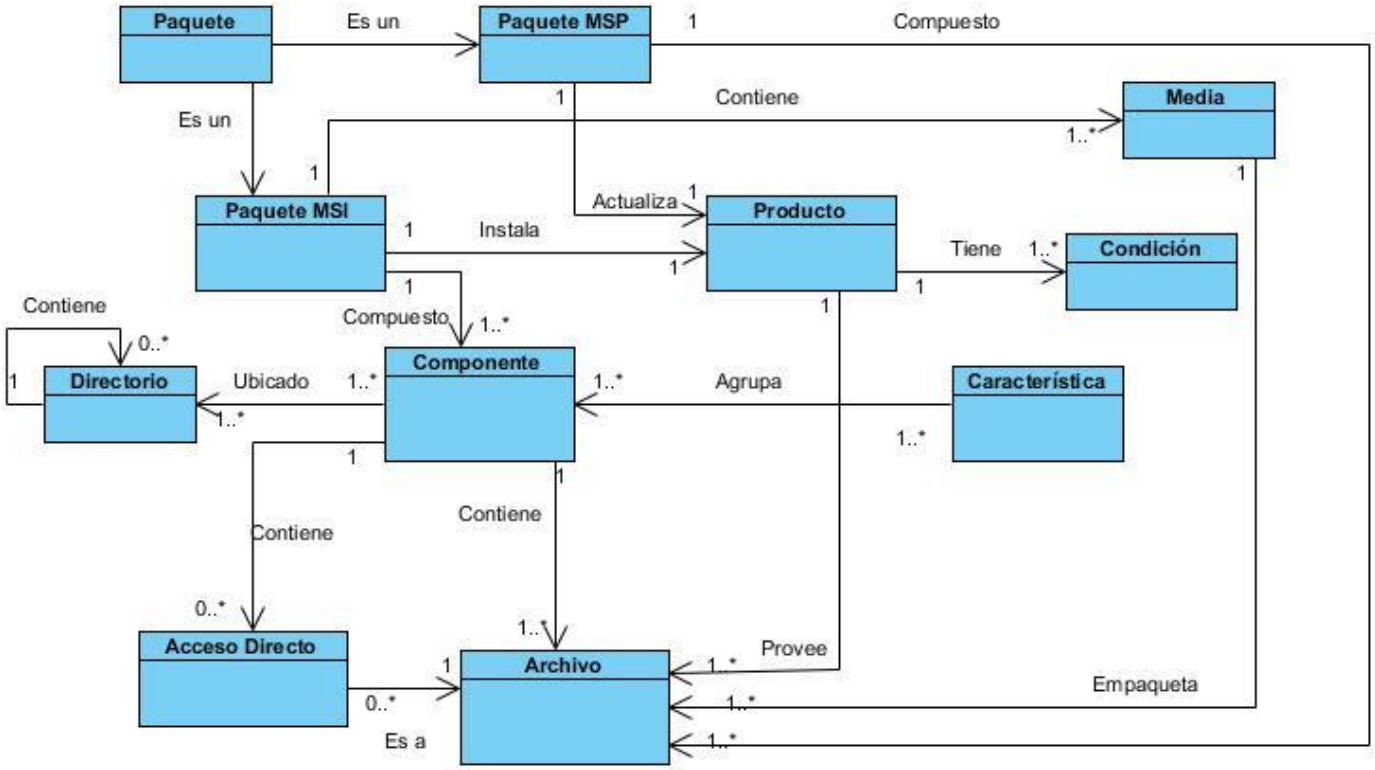

Figura 2.1. Modelo de Dominio.

2.4 Requisitos del Sistema.

Los requisitos del sistema constituyen la base para asegurar el correcto funcionamiento, presencia y por ende, aceptación final del software. Una obtención de requisitos con calidad es un pilar fundamental para el éxito del trabajo a realizar.

2.4.1 Requisitos Funcionales.

Los requisitos funcionales serán las condiciones o capacidades con que una aplicación debe cumplir, constituyen la serie de acciones que serán objeto de automatización en etapas posteriores del proceso de desarrollo del software. Estos describen las funcionalidades que se espera que el sistema cumpla para satisfacer las necesidades del usuario.

Los requisitos funcionales con los que el Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC debe cumplir son los siguientes:

RF1 Crear proyecto.

1.1 Crear proyecto de instalación MSI.

1.2 Crear proyecto de actualización MSP.

RF 2 Crear perfil del proyecto MSI.

- Nombre del producto.
- Versión del producto.
- Fabricante.
- Identificador del producto.
- Código de actualización.

RF 3 Crear perfil del proyecto MSP.

- Nombre de la actualización.
- Descripción de la actualización.
- Clasificación de la actualización
  - Actualización.
  - Actualización de Seguridad.
  - Actualización Crítica.
  - Revisión.
  - Service Pack.
- Especificar paquete MSI versión actual.
- Especificar paquete MSI versión superior.

RF 4 Guardar proyecto.

RF 5 Abrir proyecto existente.

RF 6 Adicionar componentes a instalar.

RF 7 Especificar requisitos de hardware a verificar.

7.1 Memoria RAM en *megabyte* (MB) o *gigabyte* (GB).

7.2 Espacio libre en disco duro en MB o GB.

RF 8 Especificar requisitos de software a verificar.

8.1 Sistema operativo *Microsoft Windows XP*.

8.2 *Microsoft .NET Framework* versión 1.1.

8.3 *Microsoft .NET Framework 1.1 Service Pack 1*.

8.4 Framework DMAX versión 2.0.

8.5 Servicios DMAX versión 2.0.

8.6 *Microsoft .NET Framework 2.0*.

8.7 *Framework MapInfo MapXtreme v6.7*.

RF 9 Crear nuevo requisito de software a verificar según el código del producto.

RF 10 Especificar requisitos de configuración a verificar.

10.1 Variables de entorno.

10.2 Claves de registro.

RF 11 Configurar claves de registro a crear.

RF 12 Adicionar variables de entorno a crear.

RF 13 Especificar accesos directos a crear.

13.1 Acceso directo en el Menú Inicio del sistema operativo.

13.2 Acceso directo en el Escritorio del perfil del usuario.

RF 14 Compilar proyecto.

RF 15 Generar paquete de instalación o de actualización.

2.4.2 Requisitos no Funcionales.

Los requisitos no funcionales identifican el conjunto de propiedades o cualidades que un sistema debe tener, con los requisitos no funcionales se indican las características principales que harán al producto atractivo, usable, rápido y confiable.

Los requisitos no funcionales con que el Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC debe cumplir son los siguientes:

2.4.2.1 Usabilidad.

Constituye uno de los requerimientos más importantes durante el desarrollo de software. Para la utilización de la aplicación es necesario poseer conocimientos elementales de utilización del sistema

operativo *Microsoft Windows* y aplicaciones informáticas. El sistema podrá ser utilizado por el rol encargado de crear los paquetes de instalación y actualización de las aplicaciones de escritorio del SIGESC.

2.4.2.2 Interfaz de Usuario.

La interfaz de usuario del sistema es de fácil utilización por los usuarios cumpliendo con:

- Una interfaz sencilla, agradable, legible y de fácil uso para el usuario.
- El diseño responderá a la ejecución de acciones de manera rápida, minimizando los pasos a dar en cada actividad.
- Los textos y mensajes en pantalla aparecerán en idioma español, a excepción de los mensajes de herramientas de terceros como el conjunto de herramienta WiX.
- La corrección de errores en la introducción de los datos será clara y fácil de identificar. La entrada de datos incorrectos será detectada claramente e informada al usuario.

2.4.2.3 Hardware.

Requerimientos mínimos: ordenador Pentium IV o superior y 512 MB como mínimo de memoria RAM. El espacio libre requerido en disco para la creación de paquetes de instalación o actualización dependerá de la aplicación a la que se le crearán los mismos, estará en correspondencia con los archivos a instalar.

2.4.2.4 Software.

El Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC está diseñado para trabajar sobre el sistema operativo *Microsoft Windows XP SP3*. Se necesita tener instalado el *Framework 4.0* de *.NET* para su correcto funcionamiento.

2.5 Modelo de Casos de Uso del Sistema.

Una vez identificados los requerimientos del sistema es necesario identificar los actores y casos de uso que guiarán el proceso de desarrollo del mismo.

2.5.1 Definición de los Actores del Sistema.

Un actor no es parte del sistema, es un rol de un usuario que puede intercambiar información o puede ser un recipiente pasivo de esta, representa a un ser humano, a un software, a un equipo de hardware o a una computadora que interactúa con el sistema.

A continuación se definen y describen los actores del sistema.

Actor	Justificación
Administrador de configuración	Es el encargado de realizar todas las operaciones concernientes a la creación de los paquetes de instalación y actualización de las aplicaciones de escritorio del SIGESC.

2.5.2 Diagrama de Casos de Uso.

Un diagrama de casos de uso del sistema representa gráficamente las actividades y su iteración con los actores. Los casos de uso agrupan o fragmentan funcionalidades del sistema de acuerdo a su relación y complejidad.

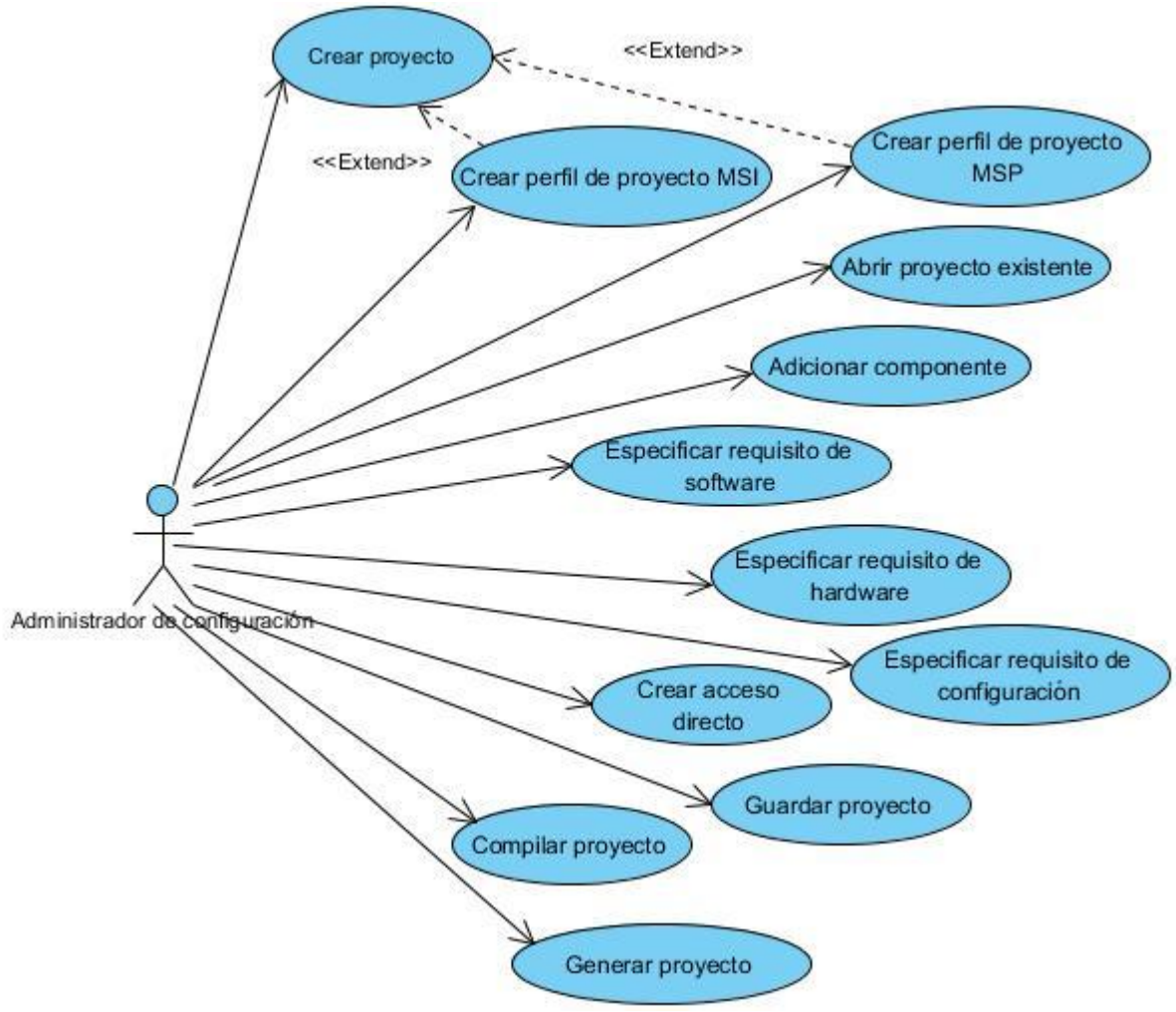

Figura 2.2 Diagrama de Casos de Uso del Sistema.

2.5.3 Descripción de los Casos de Uso del Sistema.

El objetivo principal de describir los casos de uso de un sistema es referir las funcionalidades asociadas a cada uno detallando su flujo de eventos. El diagrama de casos de uso brinda una representación gráfica de elementos importantes que los programadores necesitan comprender rápidamente pero que sin una descripción de los casos de uso sería insuficiente. La descripción detallada de cada caso de uso del sistema se encuentra en el **Anexo 1** del presente documento.

2.5.3.1 CU Crear proyecto.

CU Crear proyecto	
Propósito:	Permite crear un nuevo proyecto.
Descripción:	En este caso de uso el usuario puede crear un nuevo proyecto. Selecciona el tipo de proyecto, el nombre del mismo y la ubicación donde se guardará.
Referencias	**RF 1**

2.5.3.2 CU Crear perfil de proyecto MSI.

CU Crear perfil de proyecto MSI	
Propósito	Establecer los datos del perfil del proyecto de instalación MSI.
Descripción	En este caso de uso se muestra una interfaz para que el usuario introduzca los datos del perfil del proyecto de instalación MSI.
Referencia	**RF 2**

2.5.3.3 CU Crear perfil de proyecto MSP.

CU Crear perfil de proyecto MSP	
Propósito	Establecer los datos del perfil del proyecto de actualización MSP.
Descripción	En este caso de uso se muestra una interfaz para que el usuario introduzca los datos del perfil del proyecto de actualización MSP. Se seleccionan los dos paquetes MSI con los que se va a crear el paquete de actualización MSP.
Referencia	**RF 3**

2.5.3.4 CU Abrir proyecto existente.

CU Abrir proyecto existente	
Propósito	Abrir un proyecto existente.
Descripción	En este caso de uso el usuario puede abrir un proyecto que haya sido creado y guardado anteriormente.
Referencia	**RF 5**

2.5.3.5 CU Adicionar componente.

CU Adicionar componente	
Propósito	Adicionar los componentes necesarios para crear un paquete de instalación MSI.
Descripción	En este caso de uso el usuario agrega los componentes (archivos dll, ejecutables, imágenes, ensamblados, etc.) necesarios para crear el paquete de instalación MSI. Se crea la estructura de directorios y se le agrega a cada directorio los componentes necesarios.
Referencia	**RF 6**

2.5.3.6 CU Especificar requisito de software.

CU Especificar requisito de software	
Propósito	Especificar los requisitos de software que deben ser verificados durante el proceso de instalación.

Descripción	En este caso de uso se muestra una interfaz para que el usuario seleccione los requisitos de software que necesita la aplicación a instalar para funcionar correctamente en el equipo destino. Durante el proceso de instalación se verifican los requisitos seleccionados y se indica al usuario en caso de no encontrarse instalado alguno en el sistema operativo.
Referencia	**RF 8, RF 9**

2.5.3.7 CU Especificar requisito de hardware.

CU Especificar requisito de hardware	
Propósito	Especificar los requisitos de hardware que deben ser verificados durante el proceso de instalación.
Descripción	En este caso de uso se muestra una interfaz para que el usuario seleccione los requisitos de hardware que necesita la aplicación a instalar para funcionar correctamente en el equipo destino. Durante el proceso de instalación se verifican los requisitos seleccionados y se indica al usuario si alguno no se cumple.
Referencia	**RF 7**

2.5.3.8 CU Especificar requisito de configuración.

CU Especificar requisito de configuración	
Propósito	Especificar los requisitos de configuración que se deben crear o verificar durante el proceso de instalación.
Descripción	En este caso de uso se muestra una interfaz para que el usuario introduzca

	los requisitos de configuración que deben ser creados o verificados durante el proceso de instalación para que la aplicación a instalar pueda funcionar correctamente en el equipo destino. El sistema configura estos requisitos para que durante el proceso de instalación sean creados o verificados.
Referencia	**RF 10, RF 11, RF 12**

2.5.3.9 CU Crear acceso directo.

CU Crear acceso directo	
Propósito	Configurar los accesos directos de la aplicación a instalar.
Descripción	En este caso de uso se muestra una interfaz para que el usuario seleccione los archivos a los que se le va a crear un acceso directo, determina la ubicación donde se creará y el nombre que tendrá el acceso directo.
Referencia	**RF 13**

2.5.3.10 CU Guardar proyecto.

CU Guardar proyecto	
Propósito	Guardar un proyecto.
Descripción	En este caso de uso el usuario puede guardar un proyecto creado con las configuraciones y los datos definidos hasta el momento.
Referencia	**RF 4**

2.5.3.11 CU Compilar proyecto.

CU Compilar proyecto	
Propósito	Permite compilar el proyecto con las configuraciones realizadas hasta el momento utilizando la herramienta candle.exe de WiX para comprobar que no tenga errores.
Descripción	En este caso de uso se chequea la configuración realizada hasta el momento al proyecto para comprobar que no tenga errores y se compila utilizando la herramienta candle.exe de WiX para verificar que la configuración realizada por el usuario es consistente. Al compilar el proyecto se obtiene un archivo con extensión *wixobj* partir del cual se crea posteriormente el paquete de instalación o actualización.
Referencia	**RF 14**

2.5.3.12 CU Generar proyecto.

CU Generar proyecto	
Propósito	Generar paquete de instalación MSI o actualización MSP.
Descripción	En este caso de uso se genera el paquete de instalación MSI o de actualización MSP utilizando el conjunto de herramientas WiX.
Referencia	**RF 15**

2.6 Conclusiones.

Los artefactos obtenidos en el capítulo que se concluye describen las características de la solución a desarrollar. Ha sido sintetizado el trabajo abordado a partir de la descripción del modelo de dominio, dejando claro los requisitos funcionales y no funcionales que indican las capacidades o condiciones que el sistema debe cumplir para suplir las necesidades identificadas con anterioridad. El modelo de casos de uso obtenido guiará las actividades relativas a la implementación que harán efectivos los requisitos identificados.

Capítulo III: Construcción de la Solución Propuesta

3.1 Introducción.

El objetivo principal de este capítulo es realizar la descripción de las actividades que se llevan a cabo en la disciplina de diseño, obteniendo como resultado los artefactos para modelar el sistema. Se exhibe una descripción del estilo arquitectónico que conforma la arquitectura definida para el diseño y construcción del sistema a desarrollar, teniendo en cuenta patrones de diseño que aportan soluciones a problemas específicos para lograr un diseño eficaz en el software orientado a objetos. Se elabora el diagrama de implementación del sistema y se brinda una descripción de cada uno de los componentes que en él se interrelacionan.

3.2 Descripción de la Arquitectura.

"…La arquitectura de un sistema constituye un amplio marco que describe su forma y su estructura, sus componentes y como estos interactúan…" Jerrold Grochow.

La descripción de la arquitectura guía al equipo de desarrollo a través del ciclo de vida del sistema. Mediante los estilos y patrones arquitectónicos se logra representar un sistema de software de forma estructurada y organizada, definiendo claramente las correspondencias y responsabilidades de los elementos que lo conforman. Existen varios modelos arquitectónicos probados y reconocidos por las ventajas de su aplicabilidad y funcionamiento, el estilo arquitectónico seleccionado para organizar la implementación del Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC es el estilo N capas, quedando estructurado en las capas de Presentación, Negocio y Acceso a Datos.

Los autores James Rumbaugh, Ivar Jacobson y Grady Booch declaran este estilo como una organización jerárquica, tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior, donde las capas intermedias resultan transparentes para el resto de las capas (25).

Capa de Presentación

La capa de presentación está dividida en dos grupos de funcionalidades: **Interfaces de Usuario** y **Acciones**. Las Interfaces de Usuario presentan información a los usuarios y aceptan entradas o respuestas de estos para ser utilizadas en el sistema. Las interfaces de usuario no implementan ningún procesamiento de negocio o reglas de validación del negocio. Las Acciones a su vez representan peticiones que se les hacen al negocio desde la interfaz del usuario. Estas acciones pueden tener una interfaz de usuario asociada o no, las que tienen interfaz de usuario asociada se les denomina clase controladora de esa interfaz de usuario.

Capa de Negocio

La capa de negocio es el núcleo de la aplicación puesto que encapsula la lógica de implementación, fundamental para la automatización de los procesos del negocio a informatizar. Debe ser responsable de representar conceptos del negocio, información sobre la situación de los procesos e implementar las reglas del negocio. Reciben las peticiones del usuario y envían las respuestas tras el proceso realizado.

Capa de Acceso a Datos

La capa de acceso a datos encapsula la lógica de implementación necesaria para gestionar los datos utilizados por el negocio y abstraer así la forma en que los datos persisten o son obtenidos. Constituye el medio de acceso al almacenamiento utilizado por la aplicación, en el que se encontrarán todos los datos referentes al proceso informatizado.

Dominio

Cada sistema requiere el paso de datos entre los distintos componentes o capas que lo componen. Estos datos suelen encapsularse en entidades informativas que no son más que clases cuyo valor fundamental reside en la información que representa y no en su comportamiento en sí.

A continuación se representa gráficamente la interacción entre las capas de la arquitectura del sistema a desarrollar.

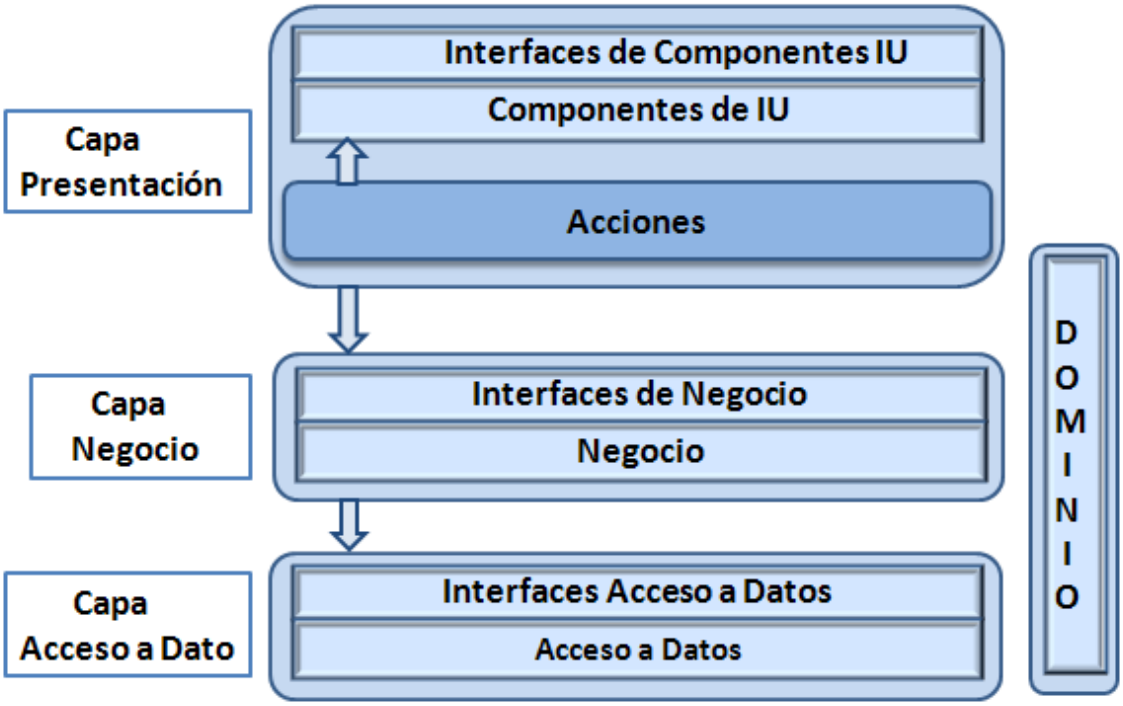

Figura 3.1 Arquitectura de la aplicación.

Como se muestra en la figura anterior el usuario interactúa con la **capa de presentación**, la cual es la encargada de crear las interfaces gráficas del sistema. Esta capa se comunica a través de **acciones** (contiene las peticiones del usuario) con la **capa de negocio,** encargada de manejar toda la lógica funcional referente al negocio del sistema, a su vez los componentes del negocio se comunican con la **capa de acceso a datos** la cual es la encargada de recuperar y almacenar toda la información que se procesa en el sistema. El **dominio** contiene las clases que representan los datos requeridos y las relaciones entre esas clases, las cuales pueden ser utilizadas desde cualquier capa del sistema.

Existen diversos mecanismos para lograr la integración entre capas y todos tienen el objetivo de resolver de una forma u otra un conjunto de problemáticas como (26):

- Lograr que las capas queden totalmente desacopladas con el objetivo de mejorar la capacidad de mantenimiento de la aplicación, facilitar el desarrollo paralelo y lograr una mayor cohesión de las funcionalidades de la capa.
- Que el desacople no provoque problemas de rendimiento.
- No provocar dependencia a otras tecnologías.
- Facilidad de uso e implementación. Si la integración entre capas requiere un esfuerzo significativo puede provocar atrasos y la posibilidad de cometer errores aumentaría considerablemente.

Los patrones de diseño forman parte de mecanismos existentes para alcanzar la integración entre las capas y resolver algunas de las problemáticas descritas anteriormente.

3.3 Patrones de Diseño.

Los patrones de diseño son un conjunto de prácticas que se utilizan para abordar problemas recurrentes en la programación orientada a objetos.

En una arquitectura orientada a objetos bien estructurada pueden estar presentes varios patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles (27).

Existen dos grupos de patrones de diseño, los patrones GRASP y los GoF.

Los patrones GRASP (en inglés *General Responsibility Asignment Software Patterns*) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Están enfocados a diseñar software robusto y reutilizables, asignando responsabilidades a los elementos de diseño y la implementación (17).

- **Experto:** Asignar una determinada responsabilidad a la clase que tiene la información necesaria para cumplirla.

- **Creador:** Asignar a una clase la responsabilidad de crear una instancia de otra clase. Se evidencia en las clases controladoras
- **Controlador:** Asignar la responsabilidad del manejo de los eventos de un sistema a una clase. Su utilización se evidencia en las clases controladoras que se encargan de la obtención y procesamiento de los datos.

Los patrones GoF (en inglés *Gang of Four*) son un conjunto de patrones de diseño creados con el objetivo de codificar y hacer reutilizables una serie de principios referentes al diseño de aplicaciones de alta calidad. Se dividen en tres grupos principales, Creacionales: Abarcan los procesos de creación de objetos, Estructurales: Tratan con la composición de las clases y objetos, De comportamiento: Caracterizan el modo en que las clases u objetos interactúan y distribuyen responsabilidades (28). Los empleados en el diseño de la solución en cuestión son:

Creacionales: solucionan problemas de creación de instancias. Ayudan a encapsular y abstraer dicha creación.

- **Singleton (Instancia Única):** Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia en el ámbito a que pertenece. Define un método estático en las clases con el fin de lograr que los objetos tengan un único punto de acceso global al admitirse una sola instancia. La clase AdministradorProyecto constituye un ejemplo del uso de este patrón, esta constituye una única instancia que representa el proyecto activo permitiendo que las demás clases tenga acceso a la información del proyecto.

Estructurales: separan la interfaz de la implementación. Se ocupan de cómo las clases y objetos se agrupan para formar estructuras más grandes.

- **Facade (Fachada):** Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un sistema. Cada capa mostrará una interfaz de acople o fachada con el conjunto de funcionalidades que oferta a la capa inmediata superior. Esto ayuda a un mayor desacoplamiento entre capas debido a que los componentes de una capa superior solo hacen referencia directa a la fachada de la capa inmediata inferior. Las clase IGestionarProyectoMSI perteneciente a la capa de negocio y IADGestionarProyectoMSI perteneciente a la capa de acceso a datos constituyen un

ejemplo donde se emplea este patrón, en estas clases se define los métodos que serán mostrados a la capa superior para ser utilizados por estas.

Comportamiento: orientados a la interacción y responsabilidades entre clases y objetos, así como los algoritmos que encapsulan.

- **Comand (Comando):** Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la petición, las clases acciones son un ejemplo del uso de este patrón.

Para lograr un mayor desacople entre las capas internas de la aplicación se utilizó la técnica basada en el Principio Inversión de Dependencia que propone (29):

- Las capas de alto nivel no deben depender de las capas de bajo nivel. Ambas capas deben depender de las abstracciones (Interfaces).
- Las abstracciones no deben depender de los detalles. Son los detalles (Implementación) los que deben depender de las abstracciones (Interfaces).

Este principio se basa en el uso de los patrones Inversión de Control (en inglés *Inversion of Control*) e Inyección de Dependencia (en inglés *Dependency Injection*).

- Patrón Inversión de Control: Delega a un componente o fuente externa la función de seleccionar un tipo de implementación concreta de las dependencias de nuestras clases.

- Patrón Inyección de Dependencia: Es realmente un caso especial del patrón Inversión de Control. Es un patrón en el que se suplen objetos y dependencias a una clase en lugar de ser la propia clase quien cree los objetos y dependencias que necesita.

Estos patrones están presentes prácticamente entre todos los objetos pertenecientes a las capas de la aplicación, permiten en cualquier momento inyectar simulaciones de comportamiento o diferentes ejecuciones reales cambiándolos en tiempo de ejecución o de configuración. Estos patrones añaden flexibilidad y conllevan a tocar el menor código posible según avanza la implementación.

Para hacer uso de estos patrones se seleccionó el *Framework Unity* implementado por *Microsoft*. *Unity* es un contenedor de dependencia extensible y ligero. Soporta inyección en el constructor, inyección de propiedad, inyección en llamadas a métodos y contenedores anidados. Es un contenedor donde se pueden registrar tipos (clases, interfaces) y también mapeos entre dichos tipos (como un mapeo de una interfaz hacia una clase), además puede instanciar bajo demanda los tipos concretos requeridos. El archivo *UnityConfiguracion.config* representa el contenedor donde están las relaciones entre los contratos y los tipos de contratos, permitiendo al patrón de Inversión de Control resolver las dependencias en tiempo de ejecución.

Unity proporciona a los desarrolladores las siguientes ventajas (29)**:**

- Permite la creación simplificada de objetos, especialmente para las estructuras jerárquicas de objetos y dependencias, lo que simplifica el código de la aplicación.
- Soporta abstracción de requerimientos, esto permite a los desarrolladores especificar dependencias en tiempo de ejecución y simplificar la gestión de preocupaciones transversales.
- Aumenta la flexibilidad al trasladar la configuración de los componentes al contenedor.
- Tiene capacidad de localización de servicios, lo que permite a los clientes almacenar o guardar en caché el contenedor de inyección.
- Permite extender las funcionalidades de los contenedores, por ejemplo; podemos implementar métodos que permitan construcciones adicionales de objetos y características de contenedores, como cache.

Gráficamente la arquitectura general de la aplicación con todos los componentes que intervienen quedaría:

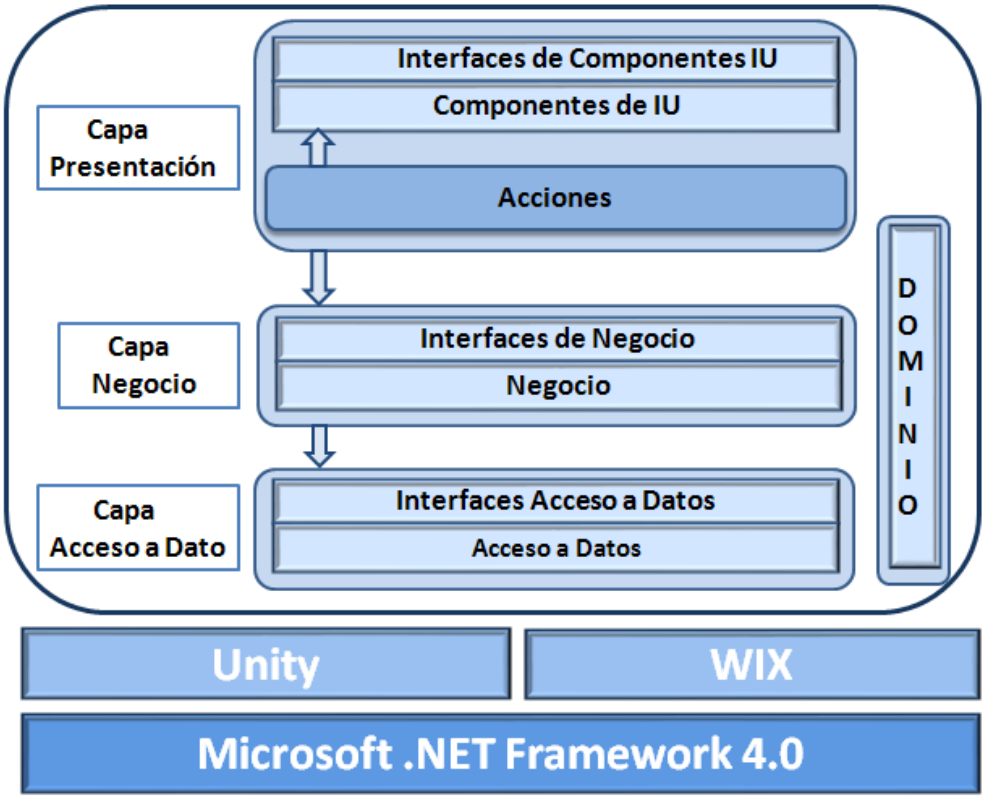

Figura 3.2 Arquitectura general del sistema.

3.4 Diseño de la Aplicación.

En el diseño se modela el sistema y se completa la arquitectura para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Previo a realizar los diagramas que muestren la solución del sistema, es necesario definir algunos conceptos que ayuden a alcanzar un producto con una calidad superior.

3.4.1 Diagrama de Clases del Diseño.

Un diagrama de clases representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Las clases fueron organizadas por colores que representan su ubicación dentro de las distintas capas arquitectónicas definidas. Cada uno de los diagramas de diseño muestra como las clases de interfaz de usuario de la capa de presentación se comunican con la capa de negocio por medio de las

acciones. A su vez, la capa de negocio se comunica con la capa de acceso a datos por medio de las clases de interfaz encargadas de lograr la comunicación entre ambas capas. Todas estas clases se relacionan con las clases del dominio, representando los objetos que conocen y manipulan los datos dentro de la aplicación.

Para una mejor comprensión de los Diagramas de Clases del Diseño (DCD) se propone la siguiente leyenda con los colores utilizados para representar las diferentes clases de la aplicación según la capa a que pertenecen.

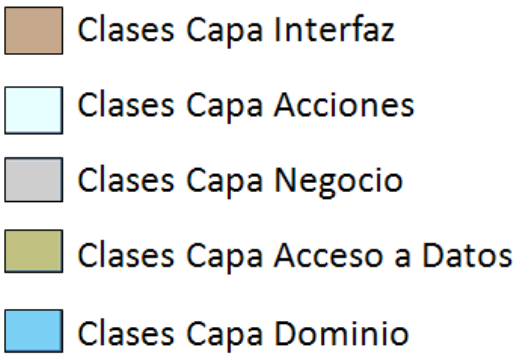

Figura 3.2 Leyenda para los DCD.

A continuación se muestran los diagramas de clases del diseño por cada caso de uso realizado. La descripción expandida de las clases del diseño del sistema se encuentra en el **Anexo 2** del presente documento.

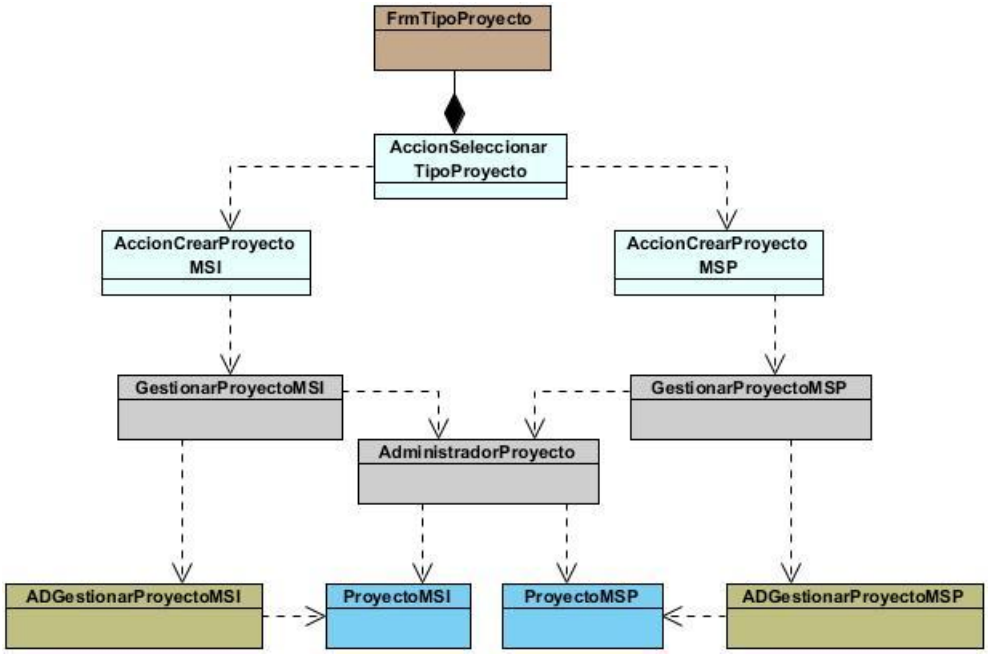

Figura 3.1 Diagrama de clases del diseño del CU Crear proyecto.

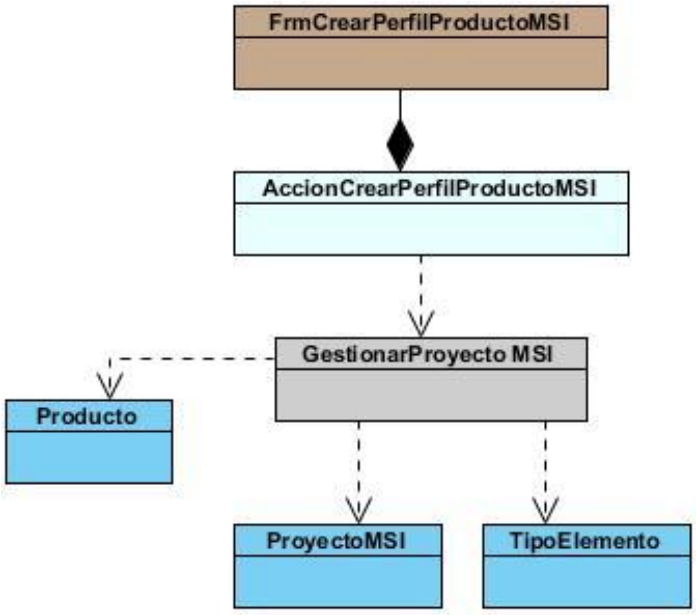

Figura 3.2 Diagrama de clases del diseño del CU Crear perfil de proyecto MSI.

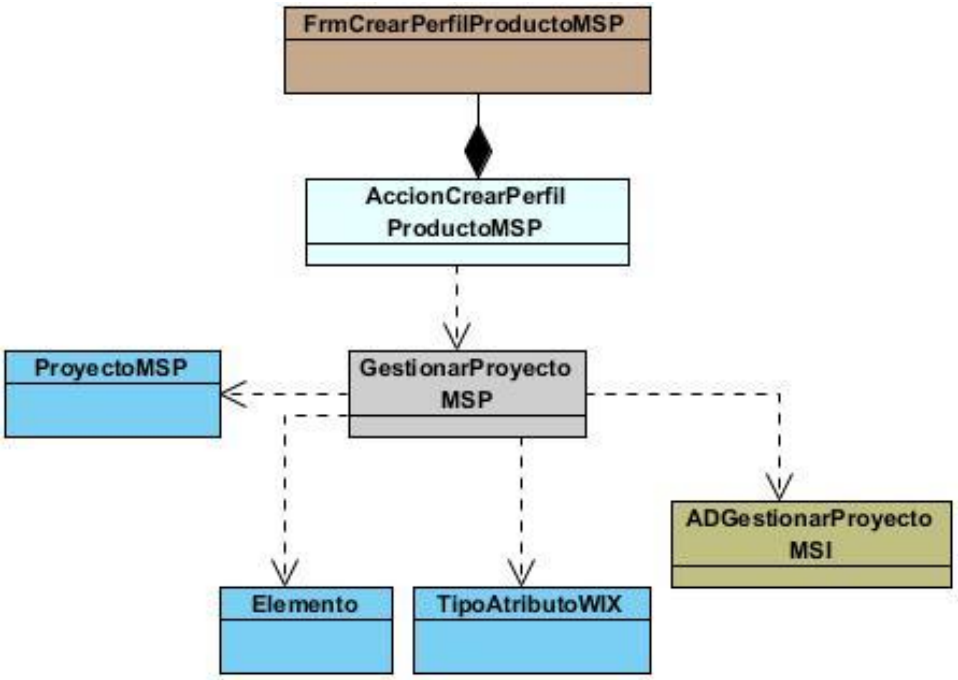

Figura 3.3 Diagrama de clases del diseño del CU Crear perfil de proyecto MSP.

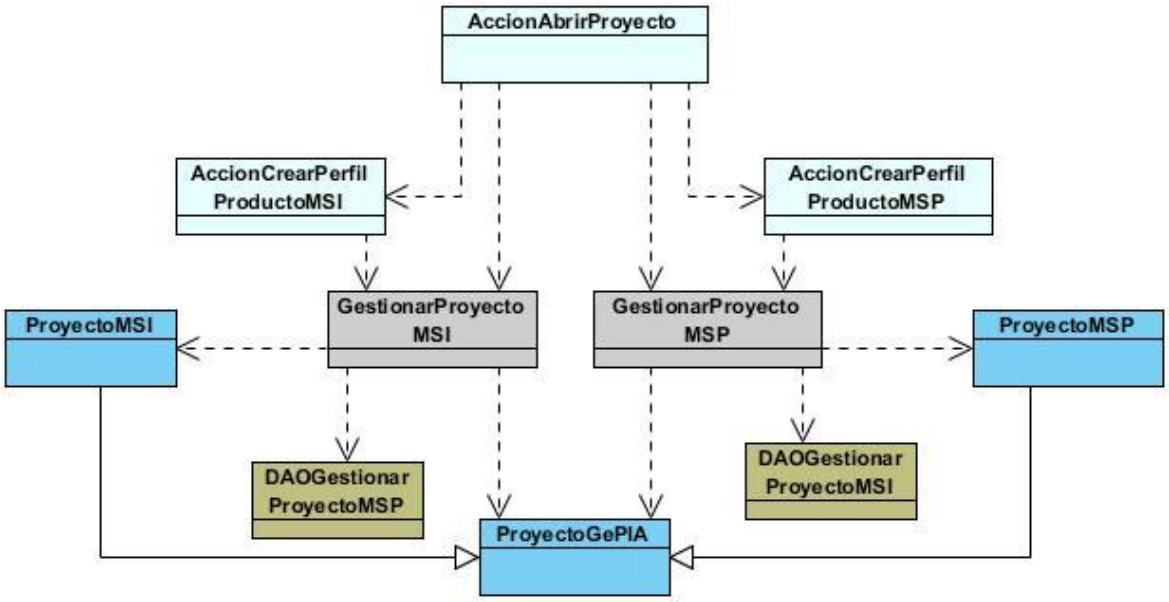

Figura 3.4 Diagrama de clases del diseño del CU Abrir proyecto existente.

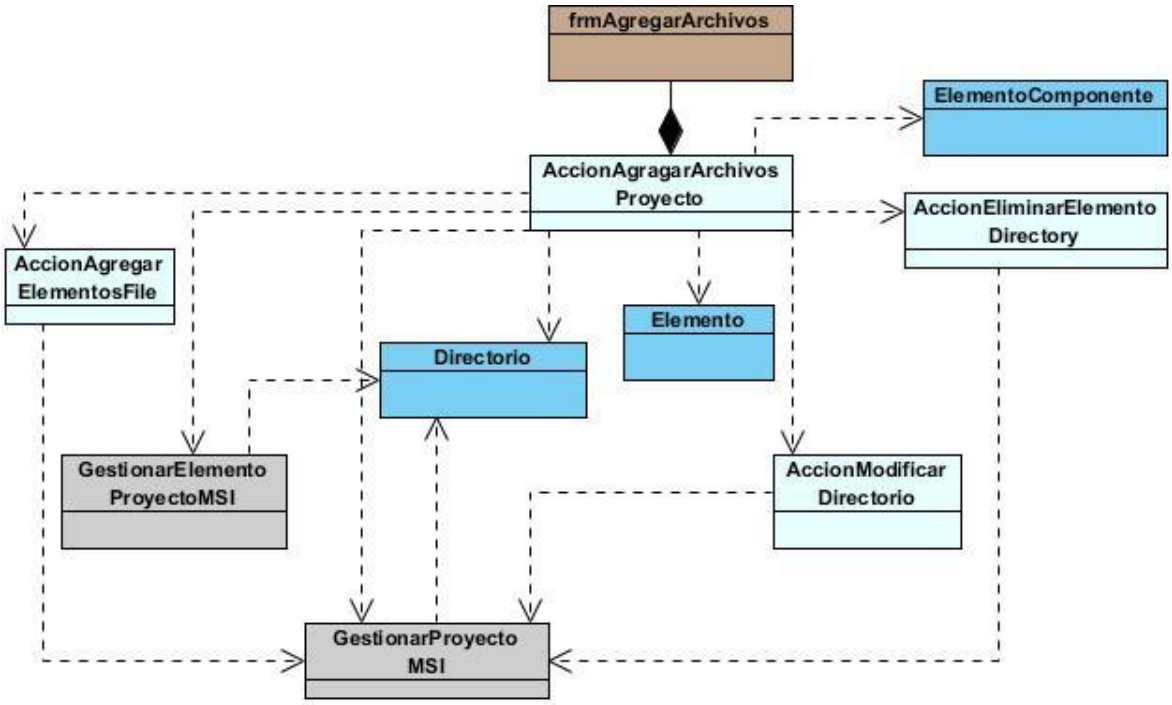

Figura 3.5 Diagrama de clases del diseño del CU Adicionar componente.

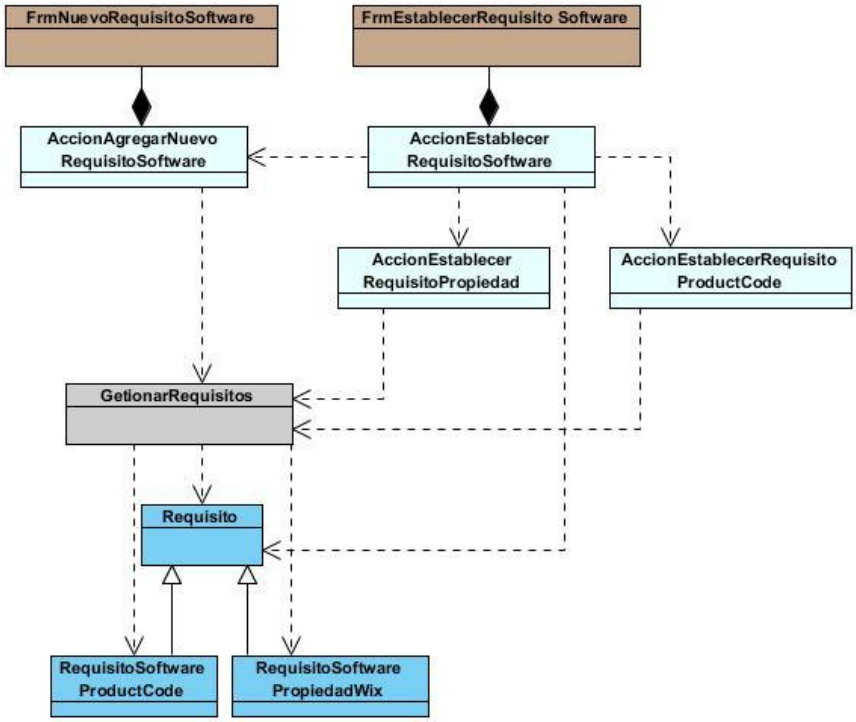

Figura 3.6 Diagrama de clases del diseño del CU Especificar requisito de software.

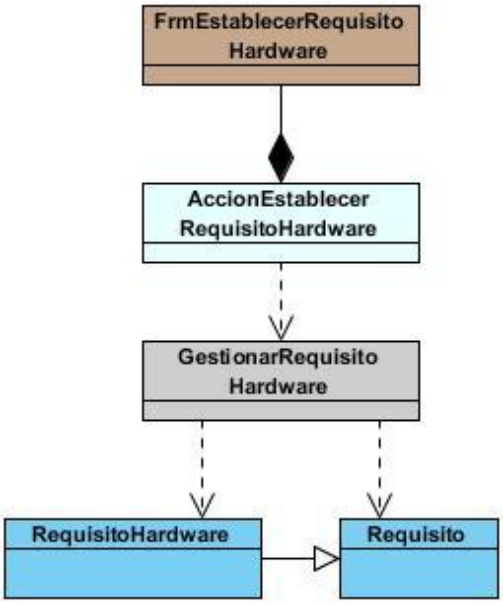

Figura 3.7 Diagrama de clases del diseño del CU Especificar requisito de hardware.

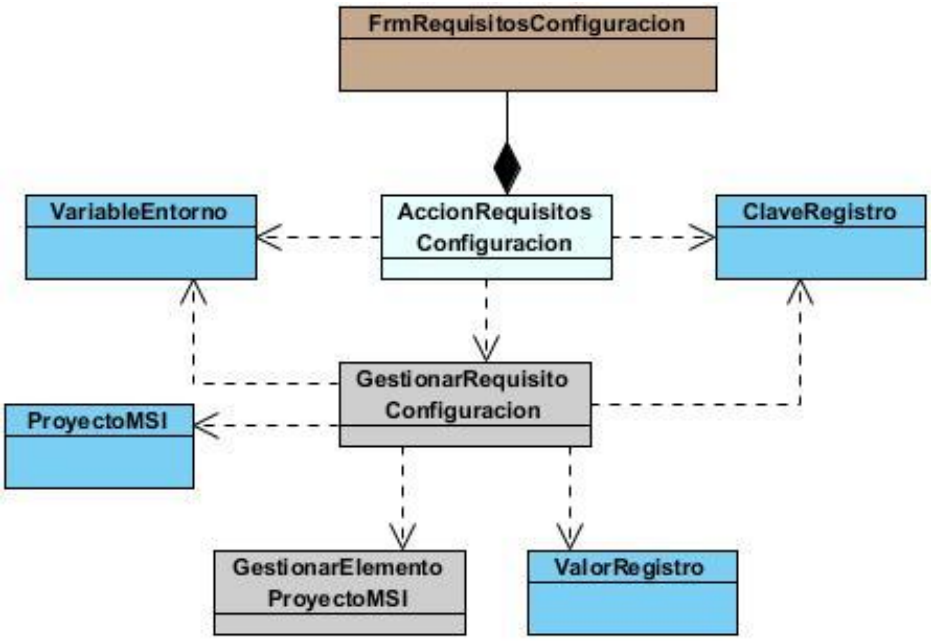

Figura 3.8 Diagrama de clases del diseño del CU Especificar requisito de configuración.

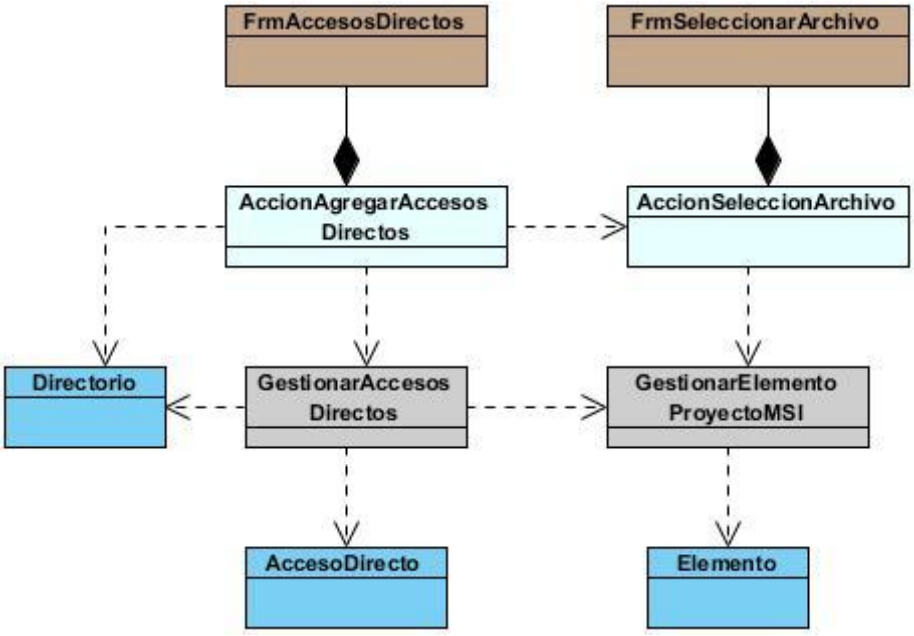

Figura 3.9 Diagrama de clases del diseño del CU Crear acceso directo.

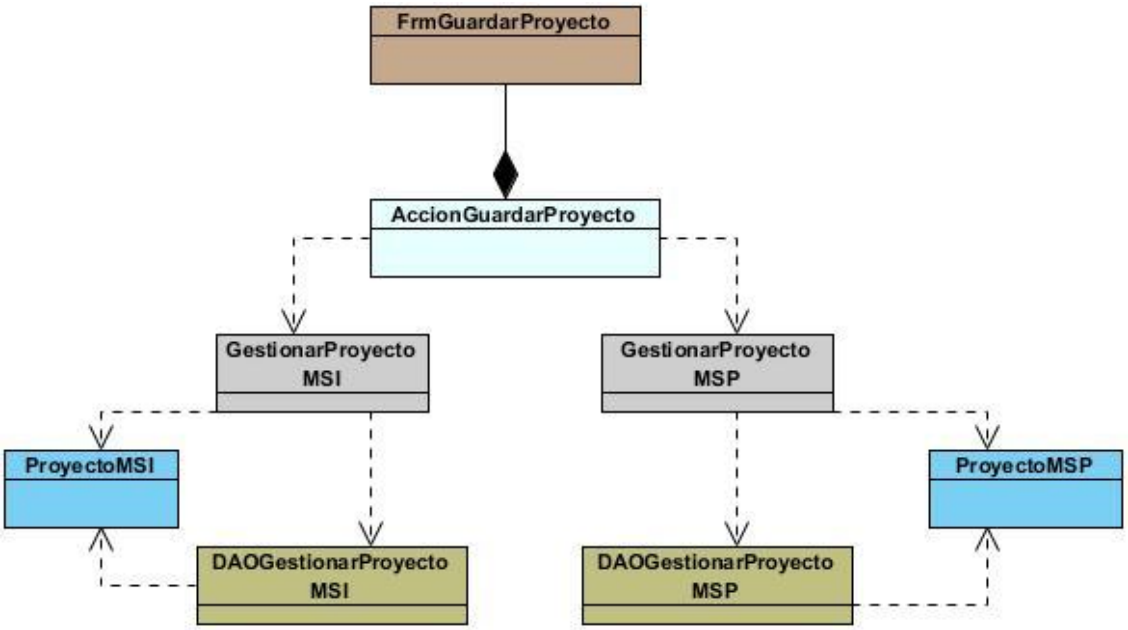

Figura 3.10 Diagrama de clases del diseño del CU Guardar proyecto.

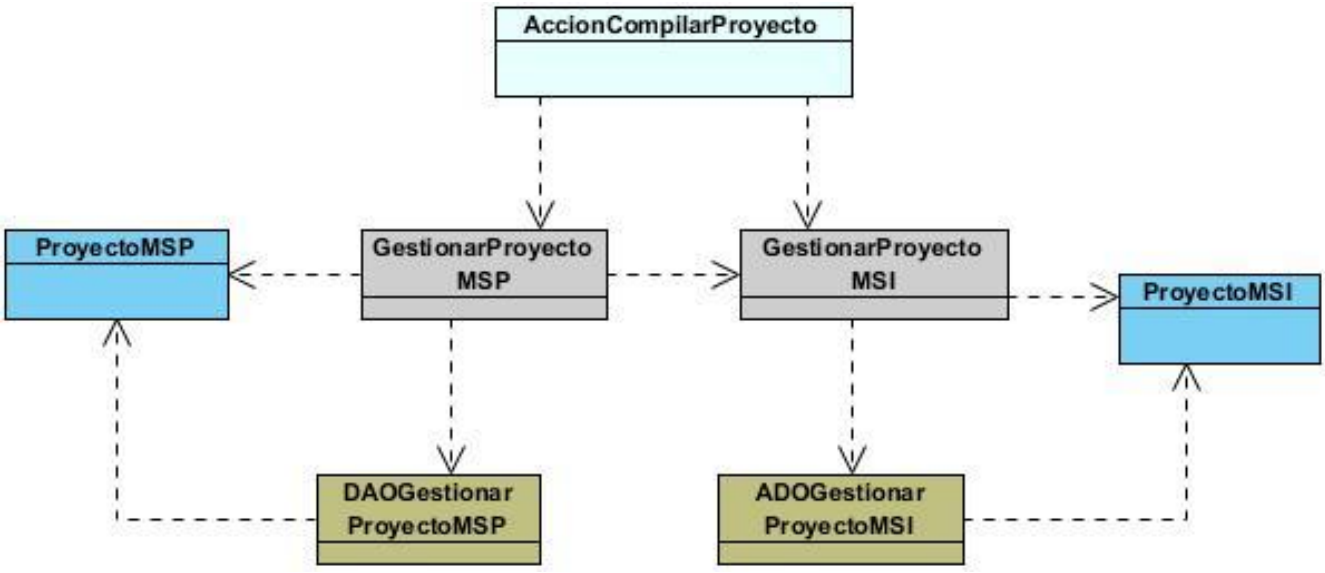

Figura 3.11 Diagrama de clases del diseño del CU Compilar proyecto.

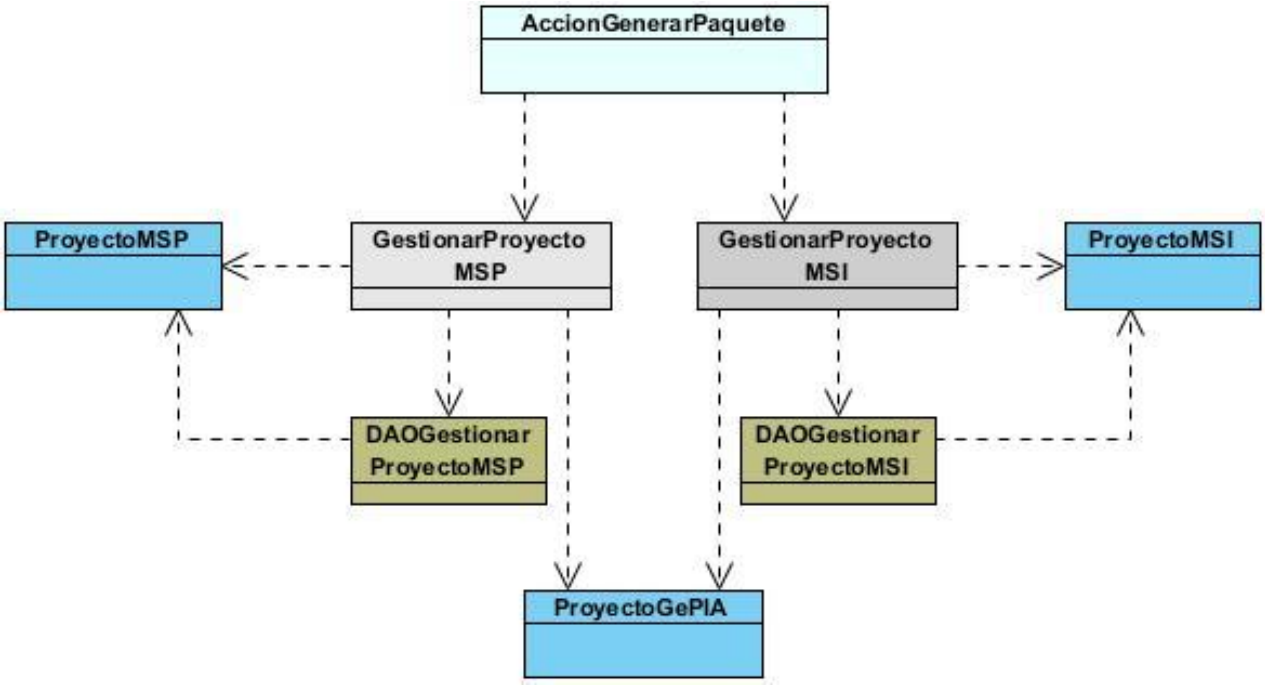

Figura 3.12 Diagrama de clases del diseño del CU Generar proyecto.

3.5 Modelo de Implementación.

El modelo de implementación describe como las clases del diseño se implementan en términos de componentes.

Los propósitos de la implementación son (25):

- Implementar las clases y subsistemas identificados durante el diseño. En particular las clases se implementan como componentes de ficheros que contienen código fuente.
- Planificar las integraciones de sistemas necesarias en cada iteración. Para ello se sigue un enfoque incremental, lo que da lugar a un sistema que se implementa en una sucesión de pasos pequeños y manejables.

- Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones de sistema.

3.5.1 Diagrama de Componentes.

El diagrama de componentes proporciona una visión física de la construcción del sistema de información. Muestra la organización de los componentes de software, sus interfaces y las dependencias entre ellos. Un componente es un módulo de software que puede ser código fuente, código binario, un ejecutable, o una librería con una interfaz definida (30).

A continuación se muestra el diagrama de componentes correspondiente al Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC.

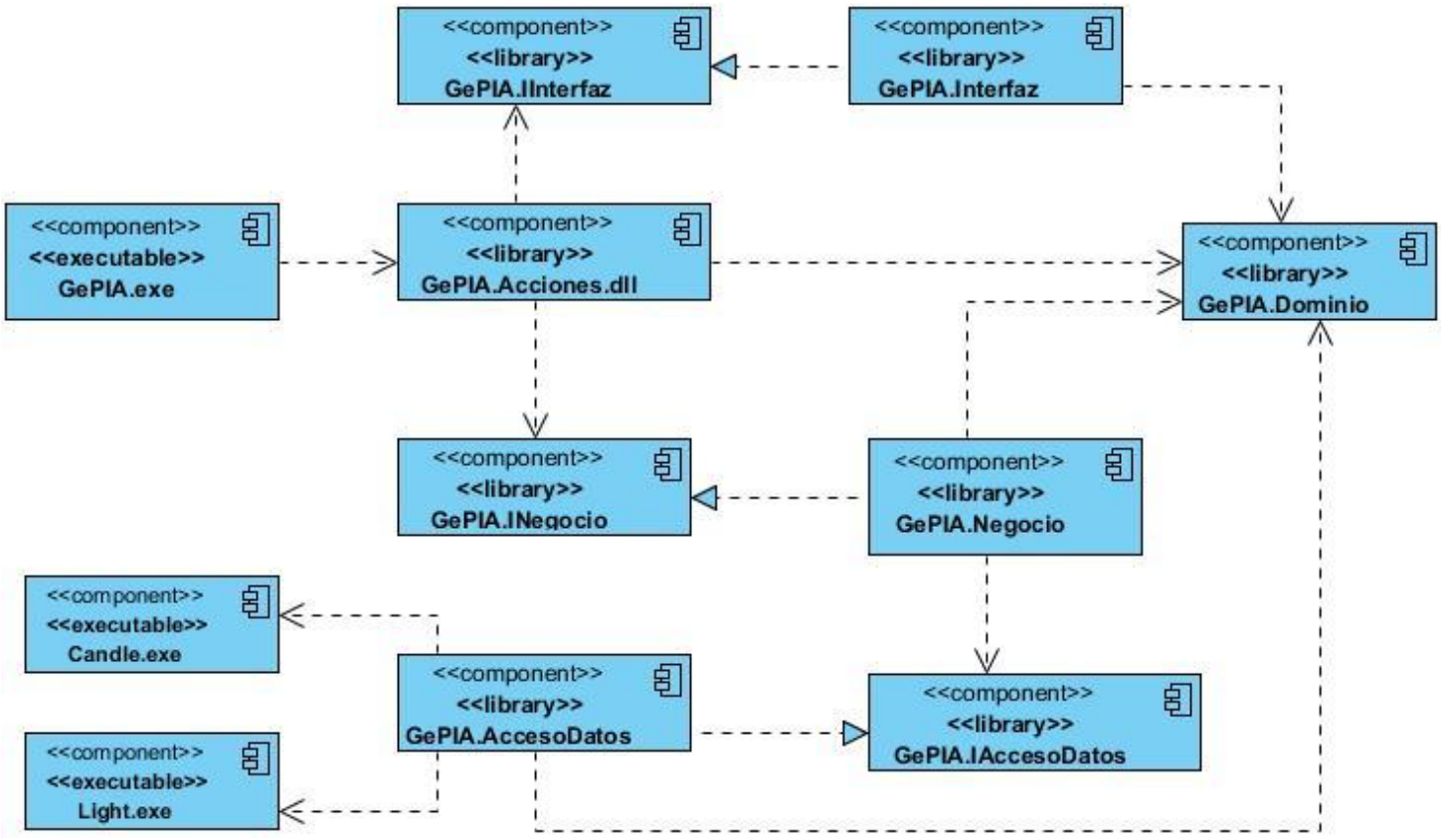

Figura 3.2 Diagrama de Componentes.

3.5.2 Descripción de los Componentes.

La descripción expandida de los componentes se encuentra en el **Anexo 3** del presente documento.

1.1 Componente GePIA.exe

Propósito: Este componente es el encargado de desencadenar la llamada a la acción que inicia la ejecución de la aplicación. Contiene los ficheros donde se definen todas las configuraciones generales de la aplicación.

1.2 Componente GePIA.IInterfaz.dll

Propósito: Este componente es el contenedor físico de las clases que definen el comportamiento de los objetos de la capa Interfaz.

1.3 Componente GePIA.Interfaz.dll

Propósito: Este componente es el contenedor físico de las clases de interfaz gráfica de usuarios.

1.4 Componente GePIA.Acciones.dll

Propósito: Este componente es el contenedor físico de las clases que realizan un flujo de acción para llevar a cabo una operación determinada. Estas clases están subscritas a los eventos que se puedan desatar en la interfaz de usuario e invocan a los métodos correspondientes de las clases de negocio.

1.5 Componente GePIA.Dominio.dll

Propósito: Este componente es el que contiene las clases entidades de la aplicación. Estas clases representan a los objetos persistentes y pueden ser utilizadas verticalmente en todas las capas de la aplicación.

1.6 Componente GePIA.INegocio.dll

Propósito: Este componente es el contenedor físico de las clases que definen el comportamiento de los objetos de la capa Negocio

1.7 Componente GePIA.Negocio.dll

Propósito: Este componente es el contenedor físico de las clases que controlan el negocio de la aplicación. En estas clases se implementan todos los algoritmos o pasos necesarios para obtener un resultado satisfactorio a partir de una petición realizada.

1.8 Componente GePIA.IAccesoDatos.dll

Propósito: Este componente es el contenedor físico de las clases que definen el comportamiento de los objetos de la capa Acceso a Datos.

1.9 Componente GePIA.AccesoDatos.dll

Propósito: Este componente es el contenedor físico de las clases encargadas de gestionar todos los datos que se manipulan en la aplicación.

1.10 Componente Candle.exe

Propósito: Este componente es el encargado de compilar el archivo XML que contiene la estructura y configuración para crear un paquete de instalación MSI o un paquete de actualización MSP.

1.11 Componente Light.exe

Propósito: Este componente es el encargado de generar los paquetes de instalación MSI o los paquetes de actualización MSP a partir del fichero XML compilado.

3.6 Conclusiones.

El estudio y modelación realizadas en capítulos anteriores permitió representar la propuesta de solución a través de los modelos de diseño e implementación, así como seleccionar un modelo arquitectónico que se

adecuara al sistema en desarrollo. Tanto esta representación como la arquitectura seleccionada y los patrones de diseño utilizados, posibilitaron implementar una solución que cumpliera con todos los requerimientos identificados como necesarios para resolver el problema planteado. La descripción de los modelos obtenidos en el presente capítulo contribuirá positivamente a la continuidad del trabajo luego de culminada esta etapa de desarrollo.

Conclusiones Generales

El Trabajo de Diploma ***"Sistema de Generación de Paquetes de Instalación y Actualización para el SIGESC"***, se estructuró en tres capítulos para darle cumplimiento al objetivo general del trabajo que consiste en: ***Desarrollar un sistema que permita generar los paquetes de instalación y actualización para las aplicaciones de escritorio del SIGESC de manera que chequeen durante el proceso de instalación los requisitos de configuración, software y hardware de este sistema.***

- La investigación aborda la temática relacionada con la instalación y actualización de aplicaciones informáticas sobre *Windows Installer*. A partir de un análisis teórico, conceptual, metodológico y práctico, se contribuye a enriquecer el conocimiento en esta esfera dando respuesta, con la solución propuesta, al problema planteado.

- El proceso de desarrollo de software ha sido guiado por la metodología RUP, los artefactos generados durante el proceso contribuirán positivamente a la continuidad de la solución obtenida en etapas posteriores de desarrollo.

- El diseño propuesto reúne los elementos apropiados para un desarrollo organizado que cumple con las necesidades descritas. A partir de la metodología de software seleccionada y la arquitectura definida, se obtuvieron los artefactos que permitieron asentar las bases para la implementación del producto. El resultado de la implementación arrojó un sistema para generar los paquetes de instalación y actualización para las aplicaciones de escritorio del SIGESC, cumpliendo con el objetivo planteado y aportando nuevas mejoras en el proceso de creación de los mismos.

Recomendaciones

Tras haber cumplido el objetivo general del presente trabajo de diploma se recomiendan los siguientes aspectos:

- Realizar las pruebas de liberación de la solución obtenida con el objetivo de verificar con un tercero el correcto funcionamiento de la aplicación.
- Incluir un mecanismo que permita adicionar nuevos requisitos de hardware a verificar por los programas de instalación generados.
- Extender la aplicación de la solución obtenida a otros proyectos de la UCI desarrollados para el sistema operativo *Microsoft Windows*.

Referencias Bibliográficas

1. **Lasso, Ivan.** Manuales y tutoriales de informática básica. *¿Qué es un instalador?* [En línea] 09 de junio de 2008. [Consultado el: 1 de diciembre de 2011.] http://www.proyectoautodidacta.com/2008/05/09/%C2%BFque-es-un-instalador/..

2. **Norton.** Norton By Symantec. *Norton By Symantec.* [En línea] 1 de agosto de 2008. [Consultado el: 4 de diciembre de 2011.] http://mx.norton.com/products/library/article.jsp?aid=vital_security#actualizaciones..

3. **Ajmani, Sameer.** *Automatic Software Upgrades for Distributed Systems.* Massachusetts : s.n., 2004.

4. **Ramirez, Nick.** *A Developer's Guide to Windows Installer XML.* Birmingham : Packt Publishing Ltd, 2010.

5. **RPM.** rpm.org. [En línea] 2007. [Consultado el: 10 de enero de 2012.] http://www.rpm.org.

6. **Bailey, Edward C.** rpm.org. *rpm.org.* [En línea] 200. [Consultado el: 10 de enero de 2012.] http://www.rpm.org/max-rpm-snapshot/ch-preface.html#S1-PREFACE-LINUX-AND-RPM-HISTORY.

7. **Dueñas, Joel Barrios.** alcancelibre.org. *alcancelibre.org.* [En línea] 2007. [Consultado el: 08 de febrero de 2012.] http://www.alcancelibre.org/staticpages/index.php/como-dpkg/print.

8. **Silva, Gustavo Noronha.** debian.org. *debian.org.* [En línea] 04 de 2003. [Consultado el: 08 de mayo de 2012.] http://www.debian.org/doc/manuals/apt-howto/apt-howto.es.txt.

9. **Microsoft Corporation.** Setup API. *msdn.microsoft.com.* [En línea] 19 de 08 de 2010. [Consultado el: 08 de enero de 2012.] http://msdn.microsoft.com/en-us/library/cc185682%28v=vs.85%29.aspx.

10. **Información sobre la tecnología de Microsoft Windows Installer**. *msdn.microsoft.com.* [En línea] 11 de 2007. [Consultado el: 08 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/h2zwd6bw%28v=vs.90%29.aspx.

11. **Setup, Inno.** Inno Setup. *jrsoftware.org.* [En línea] Inno Setup, a. [Consultado el: 03 de marzo de 2012.] http://www.jrsoftware.org/isinfo.php#features.

12. **Inno Setup Frequently Asked Questions.** [En línea] Inno Setup. [Consultado el: 03 de marzo de 2012.] http://www.jrsoftware.org/isfaq.php#msi.

13. **BitRock InstallBuilder.** InstallBuilder Features. [En línea] [Consultado el: 06 de abril de 2012.] http://installbuilder.bitrock.com/installbuilder-features.html.

14. **Microsoft Corporation.** Microsoft .NET. [En línea] 2009. [Consultado el: 23 de enero de 2012.] http://www.microsoft.com/net/overview.aspx.

15. **InstallShield.** http://www.flexerasoftware.com. *http://www.flexerasoftware.com.* [En línea] InstallShield, 20 de 01 de 2012. [Consultado el: 02 de marzo de 2012.] http://www.flexerasoftware.com/products/installshield/installshield.htm.

16. **Sanchez, María A. Mendoza.** *Metodologías de desarrollo de software.* 2004.

17. **Larman, Graig.** *UML y Patrones.Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Mexico : s.n., 2003.

18. **Sommerville, Ian.** *Ingeniería del software.* Madrid : Pearson Education.SA, 2005. pág 712.ISBN:84-7829-074-5.

19. **Microsoft Corporation.** Inicio de .NET Framework. *msdn.microsoft.com.* [En línea] [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/netframework/aa569263.

20. **Mackey, A.** *Introducing .Net 4.0 whit Visual Studio 2010.* New York, United States of America : s.n., 2010.

21. **Microsoft Corporation.** Introducción a Visual Studio 2010. *msdn.microsoft.com.* [En línea] 2010. [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/fx6bk1f4.aspx.

22. **Lo más destacado de Visual Studio 2010.** *msdn.microsoft.com.* [En línea] 2010. [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/dd547188.aspx.

23. **Troelsen, A.** *Pro C# 2010 and the .NET 4 Platform.* New York,United States of America : s.n., Fifth Edition. 1430225491.

24. **Carolina, Ivette Martínez ;.** Modelo de Dominio. [En línea] [Consultado el: 10 de febrero de 2012.] http://www.ldc.usb.ve/~martinez/cursos/ci3715/clase6_AJ2010.pdf.

25. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** El Proceso Unificado de Desarrollo del Software. [En línea] 2000. [Consultado el: 10 de Mayo de 2012.] http://bibliodoc.uci.cu/pdf/reg00060.pdf.. ISBN: 84-7829-036-2..

26. **Eddy, Sánchez Tellez.** *Especificación de la Arquitectura Base del Sistema de Gestión de Emergencia y Seguridad Ciudadana 171.* Ciudad de la Habana: Universidad de las Ciencias Informaticás : s.n., 2008.

27. **Reynoso, Carlos K.N.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

28. **Gamma, Erich, Helm, Richard y Johnson, Ralph.** *Design Patterns-Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

29. **Ramos, Miguel Angel Barroso y Calvarro, Javier Nelson.** *Guía de arquitectura N-Capas orientado al Dominio con .NET 4.0.* España : Krasis Consulting.S.L, 2010. ISBN:978-84-936696-3-8.

30. **Pressman, Roger S.** *Ingeniería de Software: Un enfoque práctico.* Nueva York, E.U.A : Editorial McGraw-Hill, 2007.

Bibliografía

1. **Lasso, Ivan. Manuales y tutoriales de informática básica.** ¿Qué es un instalador? [En línea] 09 de junio de 2008. [Consultado el: 1 de 12 de 2011.] http://www.proyectoautodidacta.com/2008/05/09/%C2%BFque-es-un-instalador/.

2. **InstallShield.** http://www.flexerasoftware.com. *http://www.flexerasoftware.com.* [En línea] InstallShield, 20 de enero de 2012. [Consultado el: 02 de febrero de 2012.] http://www.flexerasoftware.com/products/installshield/installshield.htm.

3. **Larman, Graig.** *UML y Patrones.Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Mexico : s.n., 2003.

4. **Eddy, Sánchez Tellez.** *Especificación de la Arquitectura Base del Sistema de Gestión de Emergencia y Seguridad Ciudadana 171.* Ciudad de la Habana: Universidad de las Ciencias Informaticás : s.n., 2008.

5. **Reynoso, Carlos K.N.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires: s.n, 2004.

6. **Microsoft Corporation.** Información sobre la tecnología de Microsoft Windows Installer. *msdn.microsoft.com.* [En línea] noviembre de 2007. [Consultado el: 08 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/h2zwd6bw%28v=vs.90%29.aspx.

7. **Inicio de .NET Framework.** *msdn.microsoft.com.* [En línea] [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/netframework/aa569263.

8. **Mackey, A.** *Introducing .Net 4.0 whit Visual Studio 2010.* New York, United States of America: s.n. 2010.

9. **Microsoft Corporation.** Introducción a Visual Studio 2010. *msdn.microsoft.com.* [En línea] 2010. [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/fx6bk1f4.aspx.

10.**Lo más destacado de Visual Studio 2010**. *msdn.microsoft.com.* [En línea] 2010. [Consultado el: 09 de febrero de 2012.] http://msdn.microsoft.com/es-es/library/dd547188.aspx.

11. **Troelsen, A.** *Pro C# 2010 and the .NET 4 Platform.* New York,United States of America : s.n., Fifth Edition. 1430225491.

12. **Ramos, Miguel Angel Barroso y Calvarro, Javier Nelson.** *Guía de arquitectura N-Capas orientado al Dominio con .NET 4.0.* España : Krasis Consulting.S.L, 2010. ISBN:978-84-936696-3-8.

13. **Pressman, Roger S.** *Ingeniería de Software: Un enfoque práctico.* Nueva York, E.U.A : Editorial McGraw-Hill, 2007.

14. **Gamma, Erich, Helm, Richard y Johnson, Ralph.** *Design Patterns-Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

15. **Corporation, Microsoft.** Centro de desarrolladores de Visual C#. [En línea] 2006. [Consultado el: 3 de junio de 2012.] http://msdn.microsoft.com/es-es/vcsharp/default.aspx.

16. **Ramirez, Nick .** *A Developer's Guide to Windows Installer XML.* Birmingham : Packt Publishing Ltd, 2010.

17. **Hernández , León, Alfredo , Rolando y Coello , Sayda.** *El Proceso de Investigación Científica. Hernández León, Rolando Alfredo.* Ciudad de La Habana : Editorial Universitaria del Ministerio de Educación Superior,, 2011. 978-959-16-1307-3..

18. **ALBET, Ingeniería y Sistemas.** *Descripción General del SIGESC. .* Ciudad Habana: s.n., 2009.

19. **ALBET, Ingeniería y Sistemas.** *Manual de Instalación.* Ciudad Habana : s.n., 2007. CT-SW-DR-031801.

20. **ALBET, Ingeniería y Sistemas.** *Especificaciones Suplementarias SIGESC .* Ciudad Habana : s.n., 2007. CT-SW-DR-031801. .