Universidad de las Ciencias Informáticas
Facultad 6

Título: Componente para la Protección de Software.

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

Autor:
Andris Villalón De la Cruz

Tutor:
Ing. Fernando Echemendía Tourt

Co-Tutor:
Ing. Rayner Pupo Gómez

La Habana Junio, 2013.

"Año 55 de la Revolución."

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma el presente a los ____ días del mes de _____ del año _____.

Andris Villalón De la Cruz

_____

Firma del Autor

Fernando Echemendía Tourt

_____

Firma del Tutor

Rayner Pupo Gómez

_____

Firma del Co-Tutor

DATOS DE CONTACTO

Autor:
Andris Villalón De la Cruz. (avillalon@estudiantes.uci.cu)

Estudiante perteneciente al departamento de Señales Digitales del centro GEYSED, perteneciente a la Facultad 6, específicamente al proyecto Video Vigilancia, ocupa el rol de programador.

Tutor:
Ing. Fernando Echemendía Tourt (fechemendia@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en el año 2008. Se desempeña como Desarrollador e investigador en el proyecto Video Vigilancia del departamento de Señales Digitales del Centro Geoinformática y Señales Digitales (GEySED). Imparte las asignaturas Optativas Procesamiento de Imágenes Digitales I y II.

Co-Tutor:

Ing. Rayner Pupo Gómez (rpgomez@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en el año 2011. Actualmente se desempeña como desarrollador del Módulo Visor en el Proyecto Video Vigilancia perteneciente al departamento Señales Digitales del Centro de desarrollo Geoinformática y Señales Digitales de la facultad 6.

Agradecimientos

A mi mamá, por haber hecho hasta lo imposible porque yo saliera adelante, por darme fuerzas para seguir y por haber sido mi inspiración en estos años.
A mi hermana, por haberme ayudado en todo lo que ha podido y por su apoyo incondicional.
A mi papá, porque de una u otra forma me ha ayudado a forjarme como mejor hombre.
A mi tía Oscarita, por haberme ayudado incondicionalmente en estos cinco años que estuve lejos de mi casa.
A mis sobrinitas Liennis y Lianna, por ser las cositas más lindas del mundo.
A mi primo "Manolo" por haberme ayudado en todo lo que ha podido y por seguirme la corriente en todas las locuras que se me han ocurrido.
A Ana que ha sido como una segunda madre, aunque en ocasiones tuvimos nuestras diferencias.
A Indira por haberse portado como mi hermana.
A Chacha por haber sido mi confidente y por su amistad incondicional, siempre serás mi "Mulata".
A todos mis compañeros de aula, los que están y a los que por una u otra razón no están, especialmente a: Vivi, Miguel, Osiel, Guido, Yunet, Breissy, Tamara, Maralis, Deimis, Yudy, Doina, Orisel, Leydis, Yorbenis, Osnelito y Adrián.
A los profesores del proyecto que de alguna forma me han ayudado en la realización de este trabajo, especialmente a Rayner, Reynier, Olga y Merlin.
A los profesores Rafael y Dayami, gracias por revisarme el documento de tesis sin tener compromisos.

A todos, muchas gracias....

Dedicatoria:

A todos los que de una u otra forma han hecho posible que este día se haga realidad.

Dedicado:

A mis padres, especialmente a mi mamá.
A mi hermana y a mis sobrinitas.
A Chacha, Indira y Ana.
A mi primo "Manolo".
A mi prima Belén.
A mi tía Oscarita.
A mi tía Carmen.
A toda mi familia.

Resumen

Debido al creciente desarrollo de la informática de estos tiempos, al auge que han alcanzado el internet y los sistemas de comunicación actual, y la generación de grandes volúmenes de información por parte de las entidades de cualquier sector, se han creado las condiciones necesarias para que las empresas apuesten por nuevas alternativas en sus sistemas informáticos con el fin de brindar un mejor servicio. Estos avances tecnológicos también han proporcionado vías necesarias para que personas deshonestas accedan a la información, lo que ha dado como resultado que las empresas inviertan en la búsqueda de alternativas para evitar el acceso a su información y la utilización ilegal de sus soluciones.

Esta investigación ha sido desarrollada con el objetivo de obtener un componente de protección para los productos implementados en el departamento Geoinformática y Señales Digitales, además de disminuir la copia ilegal y la piratería de software. El sistema brinda características necesarias para proteger los productos a través de una llave de licencia única generada específicamente para cada estación de trabajo. Además brinda alternativas que controlan si la licencia está activa o no, a través de la gestión de las fechas de inicio y vencimiento, devuelve el período activo de la licencia y la cantidad de días restantes, permite al usuario obtener información de la licencia, así como renovarla en caso de que esta esté inactiva.

Palabras Clave:
Componente de software, llave de licencia, seguridad, protección.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con el surgimiento de la Informática como ciencia y su aplicación en actividades humanas se concibe una nueva forma de ver el mundo, surge la llamada sociedad de la información, la cual sufre una constante y progresiva transformación que la impulsa cada vez más al desorden tecnológico. Es por ello que se hace necesaria la utilización de técnicas o métodos de protección para controlar el crecimiento y la integridad de la información.

El constante progreso de la tecnología trajo consigo que las empresas apostaran por un desarrollo técnico, al hacer grandes inversiones en sistemas informáticos que le faciliten llegar al cliente, mejorar sus procesos o facilitar sus tareas. Las aplicaciones de la Informática en este ámbito son sumamente amplias, pero esto trae implícito una serie de consecuencias adicionales como la piratería y copia ilegal de software, acciones que hoy en día está haciendo que las empresas productoras de software pierdan grandes cantidades de dinero como consecuencia de la distribución ilegal.

Tan pronto como los ordenadores empezaron a ser populares la copia ilegal de software comenzó a ser considerada un problema importante. Además, el desarrollo de las comunicaciones y la interconexión de ordenadores agudizaron el problema derivado de la copia ilegal del software al abrir una nueva brecha en las comunicaciones. En la actualidad, otras circunstancias como el perfeccionamiento de las herramientas de análisis de código y la creciente popularidad del Internet, así como las posibilidades que éste abre a la comunicación masiva de información, han ayudado a agudizar la copia ilegal del software.

Las universidades e instituciones cubanas no se encuentran ajenas al proceso de protección de software, las que cada día se esfuerzan por incorporar y mantener la seguridad en sus sistemas informáticos, es por ello que el Centro de Desarrollo Geoinformática y Señales Digitales (GEySED), perteneciente a la Facultad 6 de la Universidad de las Ciencias Informáticas (UCI) y específicamente el proyecto Video Vigilancia, desean proteger de forma eficiente y segura sus productos. De esta forma se aseguraría la integridad de las soluciones desarrolladas en el centro GEySED, además de reducir las incidencias en actividades delictivas como la copia ilegal de software y la piratería.

Actualmente, las soluciones desarrolladas en el proyecto Video Vigilancia y en otras áreas del centro GEySED se encuentran desplegadas en varias entidades de la nación. Muchas de estas soluciones no cuentan con un medio de protección que garantice la seguridad de las aplicaciones desarrolladas en el departamento Señales Digitales. Un caso muy particular es el proyecto Video Vigilancia, el mismo cuenta

con un componente que brinda protección anti copia a la versión actual del producto Suria 1.0, pero esta solución se encuentra disponible solo para sistemas *Windows*, lo que a su vez hace que el desarrollo del componente esté orientado sobre los mismos estándares, impidiendo que éste pueda ser aplicado a la segunda versión del producto Suria que actualmente se encuentra en desarrollo.

Además de estas circunstancias, la universidad a través de un contrato con la entidad que desee adquirir la nueva versión del producto Suria, no posee una alternativa para evitar que personas deshonestas incurran en delitos de piratería y copia ilegal del software. La ocurrencia de este tipo de actividades delictivas está haciendo que las empresas productoras de software busquen alternativas viables para proteger sus soluciones.

Atendiendo a la situación planteada anteriormente se formula como **problema a resolver**: ¿Cómo garantizar la protección de los productos de escritorio del departamento Señales Digitales?

Por lo que se delimita como **objeto de estudio:** El proceso de protección de software de aplicaciones de escritorio. Para dar solución al problema de la investigación científica se establece como **objetivo general:** Desarrollar un componente que garantice la protección de los productos de escritorio del Departamento Señales Digitales. Enmarcado en el **campo de acción:** Los procesos de protección de software de aplicaciones de escritorios a través de una llave de licencia única.

Por lo anteriormente planteado se define como **idea a defender**: Con la realización de un componente de protección para los productos de escritorio del departamento Señales Digitales, se garantizará la protección de las soluciones desarrolladas en dicho departamento.

Para dar cumplimiento al objetivo general de la presente investigación se proponen las siguientes tareas de investigación:

1. Caracterización de los procesos necesarios para la protección de software.
2. Caracterización de las herramientas y tecnologías necesarias para la implementación del componente.
3. Realización del diseño del componente para la protección de software.
4. Implementación del componente para la protección de software.
5. Realización de pruebas para comprobar el correcto funcionamiento del componente.

Como métodos de la investigación científica se utilizaron:

Métodos Teóricos

Análisis - Síntesis: este método permite buscar la esencia de los fenómenos, los rasgos que lo caracterizan y los distinguen. Su objetivo en una investigación es analizar las teorías, documentos, etc., permitiendo la extracción de los elementos más importantes que se relacionan con el objeto de estudio. Permitió a partir de la información y documentación estudiada sobre el tema, en diferentes fuentes, la obtención de un conocimiento general durante la investigación.

Histórico - Lógico: permite estudiar de forma analítica la trayectoria histórica real de los fenómenos, su evolución y desarrollo. Su objetivo en una investigación es constatar teóricamente cómo ha evolucionado un determinado fenómeno en un período de tiempo, en toda su trayectoria o en un fragmento temporal de la lógica de su desarrollo. Permitió realizar un análisis profundo sobre la evolución y comportamiento de los componentes para la protección de software durante las últimas décadas.

Modelación: es una reproducción simplificada de la realidad, cumple una función heurística, que permite descubrir y estudiar nuevas relaciones y cualidades del objeto de estudio. La modelación es justamente el proceso mediante el cual creamos modelos con vistas a investigar la realidad. Permitió comprender mejor el funcionamiento que debía tener el componente, así como una mejor conformación mediante la realización de los diagramas necesarios para una correcta construcción del mismo.

Métodos Empíricos

Entrevista: es una conversación planificada para obtener información. Su uso constituye un medio para el conocimiento cualitativo de los fenómenos o sobre características personales del entrevistado y puede influir en determinados aspectos de la conducta humana por lo que es importante una buena comunicación. Permitió identificar la situación problemática y la obtención de los requisitos funcionales del sistema a partir de una entrevista realizada de forma intencional al personal involucrado con dicho tema, específicamente a profesores del proyecto Video Vigilancia. (Ver Anexo 1)

El presente documento está constituido por los siguientes capítulos:

En el Capítulo I se explican los diferentes conceptos asociados al problema que permiten comprender la solución propuesta, así como las diferentes soluciones existentes en el mundo relacionadas con la protección de software especificando la inconveniencia de las mismas para ser utilizadas en las

soluciones del Departamento. Además se describen y caracterizan las herramientas a utilizar para la implementación del sistema.

En el Capítulo II se especifican los artefactos generados correspondientes a los diferentes flujos de trabajo según la metodología de desarrollado utilizada, que garantizan un mejor entendimiento y una correcta implementación del sistema propuesto.

En el Capítulo III se especifican las acciones asociadas al análisis y diseño del módulo de protección de software así como las posibles pruebas para comprobar el correcto funcionamiento del componente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción.

En este capítulo se exponen algunos principios o axiomas que rigen el funcionamiento de la protección de software y las relaciones entre los procesos que de alguna manera inciden o forman parte de la aplicación Informática que se pretende desarrollar. Se describe el estado del arte del objeto de estudio y varias de las propuestas alternativas existentes en el mercado que permiten brindar solución a la problemática identificada. Se detallan además las herramientas y tecnologías que se emplean en el desarrollo de dicho componente.

1.2. Conceptos asociados al dominio del problema.

Los conceptos asociados al problema que se explican a continuación permiten tener un dominio teórico del proceso de protección de software. Este epígrafe es de gran importancia para el desarrollo del trabajo científico; la información adquirida en la investigación servirá de base para los resultados esperados.

1.2.1. Componente.

Según el Taller Internacional sobre Programación Orientada a Componentes[1], un componente es la unidad de composición con interfaces especificadas contractualmente, con dependencias explícitas de acuerdo al contexto. Un componente de software puede ser desplegado de forma independiente y puede participar en composiciones de terceras partes (Bosch, 1998).

Por otra parte el Grupo de Gestión de Objetos[2] considera que es una unidad modular con interfaces bien definidas, que es reemplazable dentro del contexto. Así, un componente define su comportamiento en términos de interfaces provistas y requerida; y dicho componente será totalmente reemplazable por otro que cumpla con las interfaces declaradas (Lira, 2011).

Otros autores como Rick Veague considera que, un componente es un pequeño módulo de software, un conjunto de características que proporciona una cierta funcionalidad particular utilizado dentro de una aplicación. Los componentes pueden funcionar en un nivel muy bajo dentro de una aplicación, haciendo las cosas muy mundanas (OMG, 2007).

[1] Evento anual de carácter científico donde profesores universitarios, doctores, e investigadores en diversas áreas de la informática presentan sus últimos trabajos.

[2] Consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos, tales como UML, XML y CORBA.

Como se puede apreciar en las bibliografías consultadas, existen varias coincidencias en cuanto a qué es un componente, por lo que se podría definir que un componente es la unidad básica de composición, con interfaces bien definidas, reemplazables dentro de un contexto, que pueden funcionar en un nivel muy bajo dentro de una aplicación y que además será totalmente reemplazable por otro que cumpla con las interfaces declaradas.

1.2.2. Protección.

Es la acción y efecto de proteger, resguardar, defender, amparar algo o a alguien. La protección es un cuidado preventivo ante un eventual riesgo o problema. (Huerta, 2000)

Protección lógica.

Protege el software de los equipos informáticos, aplicaciones, y datos como contraseñas y datos personales, de robos y pérdida de datos. También está dada por un marco legal o normativo. En este sentido, la protección habla de mecanismos existentes para garantizar el respeto de los derechos de compradores y usuarios. (Huerta, 2000)

1.2.3. Software.

Se conoce como software al equipamiento lógico o soporte lógico de un sistema informático, comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware. (Word Reference, 2010)

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación. (Ecured, 2011)

Considerando estas definiciones, el concepto de software va más allá de los programas de computación en sus distintos estados: código fuente, binario o ejecutable; también su documentación, los datos a procesar e incluso la información de usuario forman parte del software: es decir, abarca todo lo intangible, todo lo no físico.

1.2.4. Seguridad.

El término seguridad posee múltiples usos. A grandes rasgos, puede afirmarse que este concepto que proviene del latín *securitas*[3] hace foco en la característica de seguro, es decir, realza la propiedad de algo donde no se registran peligros, daños ni riesgos. Una cosa segura es algo firme, cierto e indubitable. La seguridad, por lo tanto, puede considerarse como una certeza. (definicion.de, 2008)

Una de las acepciones del término es el que se utiliza en Informática, un concepto moderno pero sumamente importante para conservar los ordenadores y equipos relacionados en buen estado.

Se puede entender como seguridad una característica de cualquier sistema (informático o no) el cual indica que ese sistema está libre de todo peligro, daño o riesgo, y que es, en cierta manera infalible. (Huerta, 2000)

1.2.5. Llave de licencia.

Las llaves de licencia para la protección de software son sistema de seguridad basado en hardware o software, los cuales brindan protección al software contra la piratería y el uso ilegal, permitiendo el acceso a la aplicación cuando la llave está conectada al equipo. (Matas, 2006)

Como llave de licencia se considera cualquier elemento externo al software que permita evitar la copia de éste, y que vaya conectado al ordenador a través de un puerto paralelo, un puerto serie, o que vaya incorporado en el interior del ordenador (software del ordenador). De esta manera, la presencia de una llave puede imposibilitar dos cosas a los usuarios finales: primero, obtener datos y segundo, utilizar un programa. (Matas, 2006)

Por lo descrito anteriormente las llaves de licencia se podrían clasificar en dos grupos, las basadas en hardware, las cuales estarían conectadas al equipo por cualquier puerto y contenida en un dispositivo externo (Memorias Flash, Discos Duros o tarjetas SD[4]) y las basadas en software, las que estarían contenidas en el sistema, ambas con los mismos propósito, brindar protección al software que se desea proteger contra la piratería y el uso ilegal, evitando la obtención de información y la utilización del sistema.

[3] **Securitas**: de securus, exento, libre de cuidados, seguro, tranquilo, lleno de seguridad y confianza, sin recelo, es una virtud romana.

[4] **Secure Digital (SD).** Es un formato de tarjeta de memoria. Se utiliza en dispositivos portátiles tales como cámaras fotográficas digitales, teléfonos móviles e incluso en videos consolas.

1.3. Análisis del objeto de estudio.

En el presente trabajo de diploma se ha definido como **objeto de estudio**: El proceso de protección de software en aplicaciones de escritorio.

1.3.1. Descripción general del objeto de estudio

Para la industria del software, la protección de sus productos es una característica importante, no solamente en cuanto a la copia ilegal del software, sino también a la protección de los derechos de propiedad intelectual del código. La comunidad de desarrollo de software necesita herramientas y tecnologías adicionales para proteger el software de la piratería, asegurar la integridad del código y prevenir el uso no autorizado de aplicaciones de software. (SafeNet, 2006)

Un sistema de protección bastante extendido y simple consiste en la incorporación de un chequeo de claves o *passwords*[5] que habilita la instalación. En caso de no disponer de la clave adecuada el software no puede ser instalado, o bien queda con alguna funcionalidad restringida. Este sistema es muy popular en el *shareware*[6]. Pese a ello, el propio software debe contener la función de validación de claves, por lo que es posible el análisis por ingeniería inversa de tal función. Como consecuencia, son usuales tanto la aparición de generadores de claves (que el software aceptará como válidas), como la publicación de listas de claves válidas en determinados lugares de Internet. (Pimentel, 2010)

Otros muchos sistemas de protección basados exclusivamente en software utilizan técnicas como la ofuscación, reorganización y auto modificación de código. Estas técnicas proporcionan protección a corto plazo y pueden ser usadas con éxito en entornos donde el software a proteger tiene una vida muy corta, como es el caso de agentes y *applets*[7]. (Pimentel, 2010)

Las técnicas mencionadas anteriormente se consideran inseguras porque un atacante potencial, utilizando herramientas de depuración, puede llevar a cabo un análisis de las instrucciones ejecutadas en el microprocesador, así como de los datos escritos y leídos desde y hacia la memoria; y de esta manera

[5] **Passwords**: palabra secreta o código, que el usuario debe suministrar durante una sesión para demostrar que él la persona que dice ser.
[6] **Shareware**: Modalidad de distribución de software, en la que el usuario puede evaluar de forma gratuita el producto, pero con limitaciones en el tiempo de uso o en algunas de las formas de uso o con restricciones en las capacidades.
[7] **Applets**: Componente de una aplicación que se ejecuta en el contexto de otro programa, no puede ejecutarse de manera independiente, tiene que ser invocado por un programa anfitrión.

determinar cómo está protegida la aplicación que se desea resguardar. Dando como resultado la violación de la protección.

Ante el incremento de la piratería de software, ha surgido una gran variedad de métodos que tratan de reducir el problema de la protección del software. Dicha variedad va desde la combinación de métodos existentes hasta la creación de nuevas e interesantes soluciones, todas con el objetivo de buscar una solución que sea lo menos vulnerable posible. Además el auge que ha alcanzado Internet ha propiciado el aumento de las publicaciones y del intercambio de información digital entre personas o entidades. Evidentemente estos elementos constituyen ventajas irrefutables en el proceso de desarrollo de la humanidad, pero junto con toda esa información, también existe información publicada que posibilita a ciertas personas llevar a cabo acciones que atentan contra la integridad del software, como lo es el caso de la piratería de software y la utilización ilegal de los mismos.

Existe además otro denominador común entre los métodos, y es que se ha logrado identificar que la solución deseada está en el campo de las protecciones basadas en hardware. Esto se debe a que sus predecesores y no olvidados métodos basados en software, son vulnerables a los ataques; debido, en gran medida, al hecho de que son ejecutados en ambientes inseguros (arquitecturas actuales de las computadoras personales) (David Lie, 2007). Por lo que con las nuevas soluciones se trata de buscar un entorno seguro, donde sean ejecutadas la aplicaciones, con algún método basado en software. Entorno seguro que se caracteriza principalmente por no estar incluido dentro de la arquitectura original de la computadora y por permitir, en ocasiones, la ejecución de código sin tener la preocupación de que pueda ser analizado utilizando alguna herramienta de depuración. (Sean W. Smith, 2007)

En otros casos se personaliza el software para que quede ligado a una máquina determinada, por ejemplo, extrayendo información de alguno de los dispositivos hardware (disco duro, tarjeta de red, etc.) o de la configuración del sistema operativo, de forma que el software debe comprobar que se está ejecutando en la máquina adecuada, esta técnica es conocida como personalización (Pimentel, 2010).

Debido a la gran variedad de soluciones (comerciales o en proceso de investigación) y la gran variedad de mecanismos utilizados en cada una de ellas, establecer una categorización formal de estas sería muy engorroso (Tuomas, 2005). Por lo que se plantea que la utilización de la técnica de personalización en el proceso de protección de software, lograría un acercamiento más exacto a la protección que se desea implementar. Aunque esta técnica basada en software al igual que las demás puede llegar a ser susceptible o anulada, y aunque para muchos presenta inconvenientes, en cuanto a los cambios de

alguna pieza en el hardware, se plantea que es la más adecuada para aplicarla como protección a los productos del departamento GEySED. Ya que lograría reducir la duplicación ilegal de los productos liberados, obligando al usuario final a utilizarlos solo en donde estén autorizados, lo cual representa el objeto de estudio de esta investigación.

1.4. Análisis de las soluciones similares existentes.

Hace aproximadamente diez años el nuevo mercado informático, el que surgía en la pequeña y mediana empresa, se dio cuenta de que necesitaba, al igual que ya hacían las grandes empresas, proteger los programas que desarrollaban y los datos con los que trabajaban. Es a partir de entonces que empiezan a surgir empresas que se dedican, entre otras cosas, a fabricar componentes externos al ordenador para el funcionamiento de elementos internos. (Matas, 2006)

El mercado de las llaves de protección hace ya tiempo que llegó a la madurez a nivel de hardware. Tal es el caso de *Hardkey*, *SerialShield SDK*, *PC Guard for .NET* y *Activation suite* (*ACCS*). La complejidad y seguridad de los diseños electrónicos hace completamente inviable la desprotección de aplicaciones actuando sobre elementos físicos como las llaves. Desgraciadamente las protecciones existentes actualmente necesitan unos complementos software para poder ser aplicarlas a la soluciones. Es por ahí, por el software, por donde son atacadas todas las protecciones y donde se encuentran las mayores deficiencias en seguridad. (Matas, 2006)

Otros sistemas de protección, aunque no tan recientes, están basados exclusivamente en software, utilizando técnicas como la ofuscación, reorganización y automodificación de código. Como se explicó anteriormente en la descripción del objeto de estudio, consisten en técnicas de protección a corto plazo y pueden ser usadas con éxito en entornos donde el software a proteger tiene una vida muy corta, lo que no impide su estudio, tomando en cuenta sus ventajas y desventajas, para potencias las características del sistema que se quiere implementar. Ejemplo de aplicaciones que utilizan estas técnicas son: *PC Guard for .NET, PC Guard for Win32, SourceArmor .NET Obfuscator, Activation center (ACEN)*, entre otras.

Seguidamente se analizan algunas de las soluciones mencionadas anteriormente, con el objetivo de lograr una mejor comprensión del tema. Con este análisis se pretende identificar cuál o cuáles de las soluciones responden al problema de la investigación. En caso de que las soluciones estudiadas no resuelvan el problema planteado durante la investigación, el estudio de estas servirá para tomar características que regirán el desarrollo del componente para la protección de software.

1.4.1. *Hardkey*.

Hardkey puede proteger utilizando dos métodos distintos de protección, uno automático que permite proteger ejecutables ya compilados y el otro invocando bibliotecas desde el código fuente. El entorno de protección automática de ejecutables le brinda la opción de especificar qué archivos asociados a ese programa se desean manejar en forma cifrada para que la información en disco sólo pueda ser accedida por el ejecutable protegido con *Hardkey*. (HardKey, 2009)

La **Protección Automática** se instala directamente y en pocos segundos sobre el programa ejecutable ya compilado, por medio de poderosos utilitarios desarrollados para el entorno *Windows*.

La **Protección Interactiva** se implementa desde el código fuente realizando múltiples llamados a funciones disponibles mediante bibliotecas DLL[8], permitiendo que el programador proteja sus programas aprovechando al máximo las nuevas ventajas de *Hardkey*.

Una vez que la aplicación fue protegida, el ejecutable sólo correrá si detecta una llave electrónica de habilitación, asegurando que no se puedan utilizar copias no autorizadas y permitiendo, por otro lado, realizar *backups*[9] del programa sin inconveniente. (HardKey, 2009)

1.4.2. SerialShield SDK.

SerialShield SDK es una solución profesional para la protección de software, totalmente libre, todo un adelanto en el campo de la protección de software mediante dispositivos externos. Está dirigida principalmente a personas que deseen tener un control absoluto sobre sus productos. Utiliza toda la protección posible en aplicaciones basados en proyectos desarrollados en plataforma .NET, gestiona todos los aspectos para garantizar que la aplicación esté dentro del período de uso y se registra correctamente en una máquina específica. Se puede utilizar en herramientas *NET (C #, VB.NET, Delphi.NET*) y herramientas para Win32 como *Microsoft Visual Basic*, *VBA Access*, *Word VBA*, *VBA Excel*, *Microsoft Visual C + +, Borland Delphi y C + + Builder*. (IonWorx, 2010)

1.4.3. PC Guard for .NET

Software de protección profesional con un sistema de concesión de licencias para. NET Framework, solo para plataformas Windows, compatible con ambas arquitecturas (32 y 64 bits). Utilizado por las más

[8] **DLL**: Biblioteca de funciones que se puede llamar por una aplicación en tiempo de ejecución.
[9] **Backups**: es una copia de seguridad o el proceso de copia de seguridad, con el fin de que estas copias adicionales puedan utilizarse para restaurar el original después de una eventual pérdida de datos.

grandes compañías en el área de las tecnologías, como es el caso de *IBM*, *Sony*, *Google*, *HP*, *3COM, NEC, Sagem, Agfa, Siemens, BP, Alcatel, Fujitsu, Maxtor, NDS, Barclays, Tesco, Caterpillar, General Electric, Philips, Bosch, entre otras*. Es aplicable en casi cualquier escenario de protección (número de días, período, número de series, *timer*), la extensión de demostración, la activación, la licencia permanente, licencia limitada, prórroga de la licencia, la eliminación de licencias, transferencia de licencias, protección de contraseña, números de serie, filtrado de IP, funciones personalizadas, contadores personalizados, cuadros de diálogo personalizados, unidad *USB* (*Flash* / HD) de bloqueo, licencias de red. Además puede ser utilizado por cualquier persona que pueda pagar un precio bastante elevado. (SOFPRO, 2011)

1.4.4. Activation suite (ACCS)

Es un conjunto de herramientas útiles para los usuarios que deseen proteger sus aplicaciones de la copia ilegal de software, cuenta con funciones de activación incrustadas o automáticas en su sistema de gestión, destinado a la generación de códigos de activación estándar y extendida, la validación de números de serie y la generación de archivos de activación, integra el cálculo de los códigos de activación a través de conexión a Internet. Este sistema de protección es desarrollado por la misma empresa que desarrolla PC Guard (SoftPro), por lo que cuenta con las mismas desventajas de su contemporáneo. (SOFPRO, 2011)

1.4.5. Sistema de licenciamiento para Suria.

Actualmente el sistema Suria, cuenta con un componente de protección que opera extrayendo características de hardware, comprueba la existencia de la llave de licencia así como su periodo de activación, permite generar licencias de prueba por un periodo de 15 días, para que el usuario pueda probar el producto y trabaja haciendo llamadas a una biblioteca desde el código fuente.

Cuenta además con una herramienta capaz de generar archivos de licencia utilizando un algoritmo de llaves públicas y privadas RSA[10], permite la generación de un archivo de licencia a través de un código de activación, es completamente funcional sobre sistemas operativos Windows y está orientado específicamente a la versión 1.0 del producto Suria, por lo que no puedes ser aplicable a otras soluciones del departamento que actualmente están migrando a software libre.

[10] RSA: Sistema criptográfico asimétrico desarrollado por Ron Rivest, Adi Shamir y Len Adleman, del Instituto Tecnológico de Massachusetts.

El estudio realizado a los sistemas anteriormente descritos ha aportado una visión más clara de la línea a seguir durante el desarrollo del presente trabajo. Se han observado características comunes en estos sistemas que se pueden tener en cuenta durante el desarrollo futuro de un sistema similar, entre las que se destacan:

- ✓ Protección basada en elementos o características del hardware.
- ✓ El ejecutable solo correrá si detecta la llave de licencia.
- ✓ Garantizan que las aplicaciones estén dentro del período de uso.
- ✓ Permiten que no se puedan utilizar copias no autorizadas.

Así como un conjunto de elementos que no serían factibles al aplicar una de estas soluciones al problema planteado, entre los que se encuentran:

- ✓ Constante verificación de la existencia de una llave de licencia basada en hardware.
- ✓ Utilización de dispositivos externos como memorias flash para su funcionamiento.
- ✓ Están desarrolladas para el trabajo en plataforma .NET.
- ✓ Productos difíciles de conseguir, por su alto costo en el mercado internacional.

Tanto las ventajas como desventajas que ofrecen estos innovadores sistemas de protección de software servirán de guía para el desarrollo del presente trabajo. El componente para la protección de software que se quiere desarrollar tendrá como premisas para su funcionamiento las ventajas mencionadas anteriormente, lo que contribuirá a un mejoramiento del sistema. Dentro de las características con las que contará el sistema se encuentran:

- ✓ Protección basada en características del hardware que el componente utilizará para construir el archivo de licencia.
- ✓ Garantizar el período de validez de la licencia, permitiendo que la licencia generada tenga un tiempo de vida, período en el cual se podrá ejecutar la aplicación a proteger.
- ✓ El componente se podrá ejecutar en sistemas operativos Linux.
- ✓ Evitará que la aplicación a proteger pueda ser utilizada en otras estaciones de trabajo donde no haya sido instalada.
- ✓ Se podrá utilizar instanciándola desde el código fuente de la aplicación a proteger, ya que se implementara en forma de biblioteca.

1.5. Definición de las tecnologías de desarrollo.

Para lograr el desarrollo del componente para la protección de software, se hace necesario tener conocimientos de las tecnologías a las que se recurrirá para alcanzar el objetivo propuesto. Además, con fines de lograr una mayor integración con la nueva versión del producto que se está desarrollando, y por las características que presentan, se seleccionan las siguientes tecnologías de desarrollo.

1.5.1. Metodología de desarrollo.

El Proceso Unificado de Desarrollo (RUP) es un proceso de ingeniería de software que proporciona un enfoque disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo de software. Su objetivo es garantizar la producción de software de alta calidad que satisfaga las necesidades de sus usuarios finales. RUP se fundamenta en seis mejores prácticas: desarrollo iterativo, administración de requerimientos, uso de arquitectura basada en componentes, modelamiento visual, verificación continua de la calidad y administración del cambio. (Torres, 2003)

Características principales de RUP (Valencia, 2006):

- ✓ Dirigido por casos de uso

Los casos de uso representan una forma particular de ver el sistema, especifican que debe hacer el mismo para cada tipo de usuarios. A partir de los casos de uso se realiza el diseño, la implementación y las pruebas, con lo cual se puede decir que los casos de uso guían el proceso de desarrollo, además de guiar la arquitectura y a su vez esta arquitectura incide en la selección de los casos de uso.

- ✓ Centrado en la arquitectura

La arquitectura define los componentes más relevantes y conforma una visión más clara del sistema que se desea desarrollar. La arquitectura involucra los aspectos estáticos y dinámicos más significativos del sistema, está relacionada con la toma de decisiones que indican cómo tiene que ser construido el sistema y ayuda a determinar en qué orden.

- ✓ Iterativo e incremental

El proceso iterativo e incremental consta de una secuencia de iteraciones. Cada iteración aborda una parte de la funcionalidad total, pasando por todos los flujos de trabajo relevantes y refinando la arquitectura. Cada iteración se analiza cuando termina. Se puede determinar si han aparecido nuevos requisitos o han cambiado los existentes, afectando a las iteraciones siguientes. Toda la retroalimentación

de la iteración pasada permite reajustar los objetivos para las siguientes iteraciones. Se continúa con esta dinámica hasta que se haya finalizado por completo con la versión actual del producto.

¿Por qué utilizar la metodología de desarrollo RUP?

- ✓ Permite su utilización en proyectos donde las personas no cuentan con experiencia en la producción de software.
- ✓ No depende de la presencia constante del cliente.
- ✓ Permite el desarrollo de proyectos a corto y largo plazo.
- ✓ Mantiene una forma disciplinada de asignar tareas y responsabilidades entre los miembros del equipo de desarrollo, lo que facilita el trabajo, determinado quién hace determinadas actividades, cuándo lo hace y cómo lo hace.
- ✓ Su desarrollo es iterativo e incremental, lo cual posibilita que en cada flujo de trabajo se obtenga un producto con un determinado nivel, y que posibilite según los cambios que se produzcan seguir trabajando sobre esa nueva versión terminada.
- ✓ La administración de requisitos facilita una mejor comprensión y control de los requerimientos de software.
- ✓ El Modelado visual del software mediante UML garantiza la construcción de la arquitectura del software posibilitando organizar el desarrollo de la aplicación, plantear la reutilización del software, hacerlo evolucionar y reducir los riesgos.
- ✓ Permite la verificación de la calidad del software por medio de las pruebas de caja negra las cuales se concentran en los requisitos funcionales del sistema y comprueban su correcto funcionamiento.

1.5.2. Lenguaje de Modelado.

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos utilizados para modelar un diseño de software orientado a objetos. Con la utilización del lenguaje de modelado se puede llegar a obtener diseños sólidos que sirvan como guía a seguir por los analistas, desarrolladores y clientes. Logrando unificar la visión y proyección del equipo de desarrollo hacia un producto de calidad (Craig, 2008).

Lenguaje Unificado de Modelado (UML 2.0).

UML es un lenguaje gráfico para especificar, visualizar, construir y documentar los artefactos que modelan un sistema de software. UML cuenta con una notación estándar y semánticas esenciales, fue diseñado

para ser un lenguaje de modelado de propósito general, por lo que puede utilizarse para especificar la mayoría de los sistemas basados en objetos o en componentes, y para modelar aplicaciones de muy diversos dominios de aplicación y plataformas de objetos distribuidos (Fuentes, 2010).

- ✓ Permite especificar, visualizar y documentar los artefactos que son generados a lo largo del ciclo de desarrollo del componente, fundamentalmente en las fases de análisis, diseño e implementación.
- ✓ Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, lo cual posibilita que se pueda usar como lenguaje de modelado en la metodología RUP.

1.5.3. Herramienta de modelado de software.

Las herramientas CASE (Computer Aided Software Engineering), son utilizadas para realizar la representación de un producto de software, mediante diagramas que se van desarrollando durante los diferentes ciclos de vida del software. Las Herramientas CASE fueron desarrolladas para facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software (Paradigm, 2010).

Visual Paradigm 8.0.

Visual Paradigm permite modelar todos los artefactos que se obtienen a partir del análisis del negocio y el sistema. Es una herramienta multiplataforma que soporta completamente el ciclo de desarrollo de un software: análisis y diseño, construcción, pruebas y despliegue. Posibilita el modelado de base de datos, requerimientos, proceso de negocio, permite realizar todo tipo de diagramas de clases, ingeniería inversa, generar código desde diagramas y generar documentación (Paradigm, 2010).

Se tendrá en cuenta Visual Paradigm para el desarrollo del componente por las siguientes características.

- ✓ Se pueden realizar distintos tipos de diagramas, por ejemplo: Diagramas de clases, de Casos de Uso, de Secuencia y de Componentes, lo que serviría como documentación para la implementación de nuevas características al sistema.
- ✓ Permite realizar ingeniería inversa del lenguaje C++, lo que facilitaría la creación de los diagramas de clases del sistema y ahorraría tiempo en la creación de estos diagramas.
- ✓ Exportación en imágenes de los diagramas que son elaborados con formato jpg.
- ✓ Genera la documentación del sistema en formato PDF, HTML y DOC.
- ✓ Genera el código de aplicación según el modelado de la aplicación.
- ✓ Genera la documentación de lo que se está modelando.

- ✓ Disponibilidad en múltiples plataformas, lo cual facilita que la realización de diagramas de modelado no dependan del sistema operativo que se tenga instalado en el equipo de trabajo.
- ✓ Uso de un lenguaje estándar que es común a todo el equipo de desarrollo facilitando la comunicación, permitiendo la realización de diagramas y modelos durante el proceso de desarrollo.
- ✓ Licencia: gratuita y comercial, fácil de instalar y actualizar.

1.5.4. Entorno integrado de desarrollo.

Un entorno integrado de desarrollo (en inglés Integrated Development Environment o IDE) es un programa compuesto por una serie de herramientas que utilizan los programadores para escribir códigos. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios de estos. Las herramientas que normalmente componen un entorno integrado de desarrollo son las siguientes: un editor de texto, un compilador, un intérprete, unas herramientas para la automatización, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones (Oviedo, 2009).

QtCreator 2.6.

Es un entorno integrado de desarrollo multiplataforma, adaptado a las necesidades de los desarrolladores, permite crear aplicaciones de escritorio y plataformas de dispositivos móviles. Sus características más relevantes se detallan a continuación (Nokia, 2010):

- ✓ Editor de código sofisticado: ofrece soporte para la edición en el lenguaje C++, ayuda sensible al contexto, completamiento de código y buena navegación.
- ✓ Diseño de interfaz de usuario integrado: proporciona dos editores visuales integrados: QtDesigner para la creación de interfaces de usuario.
- ✓ Proyecto y Gestión de Construcción: si se importa un proyecto existente o se crea uno desde cero, genera todos los archivos necesarios.
- ✓ Escritorio y objetos móviles: ofrece soporte para crear y ejecutar aplicaciones para equipos de escritorio.

1.5.5. Marco de Trabajo.

En el desarrollo de software, un marco de trabajo o framework (en inglés) es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede

incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto**.** (CodeBox, 2011)

Qt 4.8.

Para el desarrollo de la presente investigación se tendrá en cuenta el framework Qt en su versión 4.8 ya que es un marco de trabajo multiplataforma para el desarrollo de aplicaciones con interfaz gráfica de usuario o de consola. Cuenta con abundante documentación, posee una arquitectura que soporta el uso de Plugins[11] y hasta el momento ha sido liberado bajo dos licencias, la LGPL y la comercial, además presenta un alto rendimiento en cualquier plataforma de trabajo. Presenta varios componentes que facilitan el trabajo a los desarrolladores, los cuales se mencionan a continuación:

- ✓ Las bibliotecas Qt: clases escritas en C++ que facilitan el desarrollo de casi cualquier aplicación de software.
- ✓ QtDesigner: potente herramienta para el desarrollo de interfaces visuales.
- ✓ QtAssistant: acceso rápido a la documentación en caso de presentar alguna duda durante el desarrollo del componente.
- ✓ qmake: simplifica el proceso de construcción de proyectos en las diferentes plataformas soportadas (Techerald, 2012).

1.5.6. Lenguaje de Programación.

Se define como lenguaje de programación al elemento dentro de la Informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que se pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes (deficinion.org, 2012).

C++

Es un lenguaje de programación basado en C que soporta directamente conceptos de la Orientación a Objetos. Desde el punto de vista de la programación orientada a objetos ofrece una buena cantidad de recursos, como son las formas de encapsulamiento, la sobrecarga de operadores y funciones, la herencia y el polimorfismo. Los mayores beneficios se aprecian a largo plazo: alta calidad del software, mayor reutilización de código y mayor facilidad de adaptación, esta última apreciada sobre todo en grandes

[11] **Plugins**: Complemento, es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.

proyectos (Katrib, 2008). Es uno de los lenguajes de programación más empleados en la actualidad. Puede emplearse mediante programación basada en eventos para crear programas que usen interfaz gráfica de usuario.

Se tendrá en cuenta para el desarrolla de la presente investigación ya que:

- ✓ Posee una alta difusión al ser uno de los lenguajes más empleados en la actualidad, cuenta con un gran número de usuarios y existe una amplia documentación, que servirá para un mejor entendimiento en caso de presentarse algún inconveniente durante el desarrollo del componente.
- ✓ Posee una gran versatilidad al ser un lenguaje de propósito general, por lo que se puede emplear para resolver casi cualquier tipo de problema.
- ✓ Se pueden realizar aplicaciones distribuibles sin necesidad de pagar una licencia, lo cual posibilita la realización de la aplicación en su totalidad sin tener que pagar por la utilización de este lenguaje.
- ✓ Presenta la facilidad de que es portable, propiciando que el código del componente para la protección de software pueda ser compilado en cualquier sistema operativo, siempre y cuando se hallan creado las bases para ello.
- ✓ Proporciona facilidades para utilizar código o bibliotecas existentes además de ser uno de los lenguajes más rápidos en cuanto a ejecución, ventaja que puede ser aprovechada en el proceso de lectura del hardware.

1.5.7. Biblioteca.

Una biblioteca es una colección o conjunto de subprogramas usados para desarrollar software. En general, las bibliotecas no son ejecutables, pero sí pueden ser usadas por ejecutables que las necesitan para poder funcionar correctamente. La mayoría de los sistemas operativos proveen bibliotecas que implementan la mayoría de los servicios del sistema. Dichas bibliotecas contienen comodidades que las aplicaciones modernas esperan que un sistema operativo provea (ALEGSA, 2011).

Libqca

La biblioteca QCA tiene como objetivo proporcionar una plataforma criptográfica haciendo uso de Plugins para el cifrado de información, utilizando los tipos de datos Qt y las conversiones de datos. Además la[12] de la implementación, mediante el uso de Plugins conocidos como proveedores. La ventaja que proporciona

[12] **API**: Application *Programming Interface* o Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos o métodos, en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

este modelo es que permite a las aplicaciones evitar las dependencias de una librería criptográfica en particular. Esto permite cambiar o actualizar fácilmente las implementaciones criptográficas. Además su funcionamiento es efectivo en sistemas operativos Linux y Windows.

La biblioteca Libqca se tendrá en cuenta para el desarrollo de la presente investigación ya que:

- ✓ Permite el fácil uso en las aplicaciones, al tener solo que incluirla como una librería más en las aplicaciones QT.
- ✓ Permite la utilización de algoritmos asimétricos de cifrado del tipo RSA.
- ✓ Cuenta con una amplia documentación que permiten la aclaración de dudas que puedan surgir durante el desarrollo del componente, así como una mejor comprensión del funcionamiento de la propia biblioteca.
- ✓ Tiene implementados los algoritmos de funciones resumen como el SHA[13] y el MD5[14].
- ✓ Permite la utilización de algoritmos AES para el cifrado de información.
- ✓ Se encuentra disponible en los repositorios de Ubuntu y se puede descargar fácilmente desde internet.

1.6. Conclusiones

Con el estudio de las soluciones similares existentes en el mundo y los conceptos asociados a la protección de software, se ha obtenido una visión más clara de los elementos y características que se deben tener en cuenta durante el diseño y construcción de un componente para la protección de software. Por lo que se llega a las siguientes conclusiones:

- ✓ La descripción general del objeto de estudio ayudo a comprender cuáles fueron las primeras técnicas de protección utilizadas. Además de evidenciar cuál es la técnica que por sus características convendría utilizar en el desarrollo del componente para la protección de software.
- ✓ La investigación realizada a las soluciones existentes de componentes para la protección de software permitió concluir que estos no podían ser utilizados como solución al problema por las desventajas mencionadas durante el análisis de las mismas ya que en su mayoría presentan un elevado costo y están destinados a productos específicos. Los mismos también cuentan con características comunes

[13] **SHA**: *Secure Hash Algorithm o* Algoritmo de *Hash* Seguro. Sistema de funciones criptográficas.
[14] **MD5**: *Message Digest Algorithm 5 o* Algoritmo de Resumen del Mensaje 5, es un algoritmo de reducción criptográfico de 128 bits ampliamente usado.

que se pueden tener en cuenta en la implementación del componente, enfocándolas en las necesidades actuales del proyecto Video Vigilancia y el departamento GEySED.

- ✓ Se caracterizaron las herramientas y tecnologías a utilizar en el desarrollo de la aplicación enunciándose de cada una de ellas las principales características y las facilidades que ofrecen al equipo de desarrollo, favoreciendo la posible integración con sistemas que lo soporten y garantizando futuras actualizaciones del sistema.

CAPÍTULO 2: DESCRIPCIÓN Y ANÁLISIS DEL COMPONENTE PARA LA PROTECCIÓN DE SOFTWARE.

2.1. Introducción.

En el siguiente capítulo se realiza una descripción del diseño del componente para la protección de software propuesto para dar solución al problema que dio origen a la presente investigación. Se define además el modelo del dominio, la descripción de las clases que lo componen, los requisitos funcionales y los no funcionales, así como casos de usos y actores involucrados, se describen los casos de usos críticos para la mejor comprensión de las funcionalidades, para así, partiendo del análisis, llegar a un sistema que cumpla con los requisitos funcionales que respondan a las necesidades existentes.

2.2. Modelo de Dominio.

Debido a que no se puede precisar la estructura de los procesos de negocio, se realiza un modelo de dominio. El modelo de dominio es el punto de partida para lograr el diseño adecuado del sistema que se desea desarrollar ya que expresa el razonamiento obtenido del problema, permite mostrar al usuario los principales conceptos que se manejan en el dominio del problema de manera visual, logrando facilitar la comprensión del sistema. Es una representación de las clases conceptuales del mundo real, no de componentes software y emplea un glosario de términos para lograr una mejor concepción de los conceptos asociado (Loja, 2008).

2.2.1. Descripción del flujo del diagrama del modelo del dominio.

El **departamento Señales Digitales** cuenta con disímiles **proyectos productivos**, muchos de estos llevan a cabo el desarrollo de varias **aplicaciones de escritorio**, ya sea por solicitud de un cliente o por necesidad del propio departamento. Las aplicaciones desarrolladas en los proyectos se encuentran desplegadas en varias **empresas** del país como parte del proceso de informatización de la sociedad. Estos productos una vez que son desplegados no cuentan con un **medio de protección** que garantice la utilización de los mismos solo en las entidades que poseen permiso de uso, lo que trae como consecuencias que **personas deshonestas** incurran en delitos como la copia ilegal y la piratería de software. La siguiente figura muestra de forma gráfica la descripción del diagrama de clases de dominio, la cual servirá de punto de partida para lograr el correcto diseño del sistema.

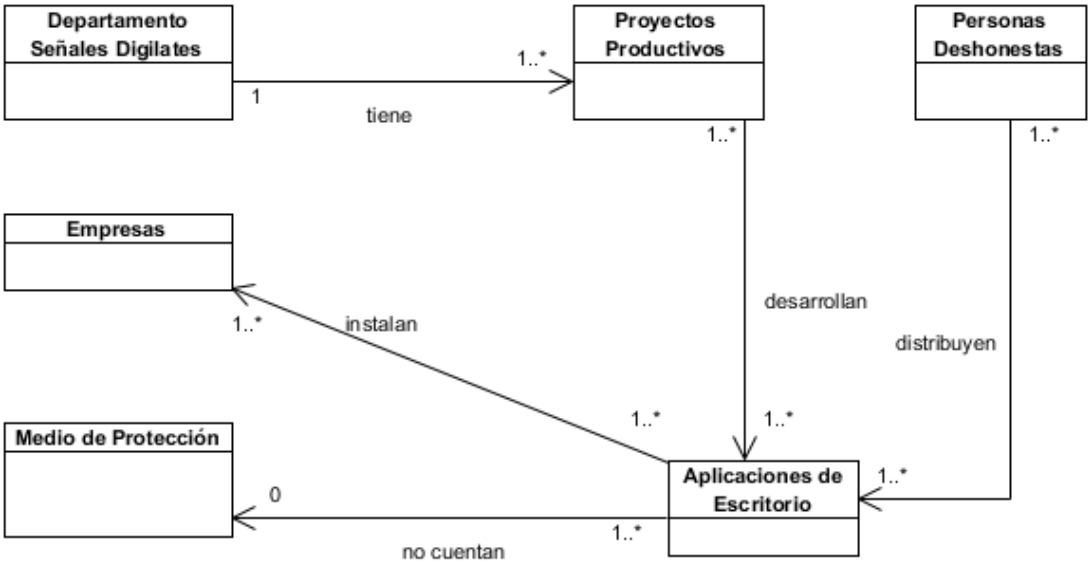

Figura 1. Diagrama de clases del dominio.

2.2.2. Descripción de las clases del modelo de dominio.

Departamento Señales Digitales: departamento perteneciente al centro de desarrollo GEySED, el mismo está compuesto por proyectos productivos como es el caso de Video Vigilancia, el Sistema Gestor de Procesos de Media, entre otros. Se encarga de gestionar todo lo referentes a sus proyectos productivos.

Proyectos Productivos: los proyectos productivos son áreas del departamento más pequeñas dedicadas específicamente al desarrollo de algún producto en específico.

Aplicaciones de escritorio: son los resultados o productos obtenidos como parte del proceso de desarrollo de software que se lleva a cabo en cada proyecto productivo.

Empresas: son las que solicitan los productos desarrollados en el departamento, donde se despliegan las soluciones obtenidas en los proyectos.

Medio de Protección: es cualquier alternativa que permita proteger al software de alguna manera, ya sea contra la copia ilegal, la piratería o para mantener la propia integridad y disponibilidad del sistema.

Personas deshonestas: personas que pueden incurrir en acciones delictivas como la piratería y la copia ilegal del software de forma intencional o accidentalmente.

2.2.3. Propuesta del sistema a desarrollar.

La solución que se plantea es la de desarrollar un componente para la protección de software que permita realizar las tareas de protección al Sistema de Video Vigilancia o cualquier otro sistema que se desee proteger, para evitar la utilización ilegal de los sistemas en ambientes no autorizados.

Este componente estará compuesto por dos aplicaciones, la primera será una herramienta con interfaz visual que se encargará de generar el archivo de licencia a partir de un código de activación, el nombre del producto que se desea proteger, el nombre del propietario de la licencia, la fecha de inicio y la fecha de vencimiento de la licencia. Con esta información la herramienta deberá ser capaz de generar el archivo de licencia además de cifrarlo utilizando técnicas de criptografía para lograr una mayor seguridad.

La segunda herramienta será una biblioteca que podrá ser invocada desde el código del sistema a proteger, la misma se encargará de brindar las funcionalidades necesarias para que el sistema cliente pueda aplicar la protección deseada, el sistema solo brindará funcionalidades para la protección, la forma de reaccionar ante cualquier incidente será responsabilidad íntegra del cliente. Esta biblioteca deberá utilizar la misma técnica de criptografía utilizada por el generador de licencias para poder descifrar la información contenida en el archivo de licencia.

2.3. Requisitos.

En el presente epígrafe se abordara acerca de los requisitos funcionales y no funcionales para lograr el correcto funcionamiento del componente para la protección de Software. Los requisitos no son más que las condiciones necesarias que debe cumplir el sistema para alcanzar los objetivos trazados.

2.3.1. Requisitos Funcionales.

Los requisitos funcionales son características conforme cada sistema, estos describen las características y funcionalidades que debe cumplir el mismo, son una condición necesaria para solucionar una dificultad y así llegar al objetivo propuesto detallando qué se debería implementar (Pressman, 2010). El componente para la protección de software tendrá como premisas los siguientes requisitos funcionales:

- ***RF 1. Obtener información del hardware.***

Este requisito permitirá al sistema hacer una búsqueda exhaustiva de toda la información referente al hardware de la PC[15] en la que se está ejecutando el componente, luego la información obtenida será

[15] **PC**: Computadora Personal o Personal Computer por sus siglas en inglés.

clasificada con el objetivo de obtener información única de los dispositivos de hardware, conformando un empaquetado solo con la información de utilidad.

- ***RF 2. Generar código de activación.***

El código de activación será el resultado de aplicar una función hash (HSA-1, HSA-2 o MD5) a la información del hardware obtenida en el requisito anterior, dicho código tendrá una longitud de 32 caracteres, los cuales se utilizarán íntegramente para la confección de la llave de licencia, este código de activación será único para cada computadora, ya que no es posible que dos computadoras presente las mismas características en el hardware.

- ***RF 3. Comprobar existencia del archivo de licencia.***

Este requisito será el encargado de velar por la existencia del archivo de licencia, además velará por que el archivo de licencia tenga un nombre específico, el cual el sistema comprobara cada vez que se ejecute.

- ***RF 4. Comprobar período activo de la llave de licencia.***

El sistema, una vez ejecutado se encargará de chequear la llave de licencia, y específicamente el período de validez de la licencia, es decir, este requisito es el encargado de velar por que la licencia este en su período activo, entiéndase período activo como el tiempo establecido para su uso.

- ***RF 5. Comprobar códigos de activación.***

Este requisito se encargará de validar que la licencia sea la adecuada para el hardware donde se ejecutó, el mismo realiza una comparación del código de activación que se encuentra en el archivo de licencia con un código de activación que obtiene en tiempo de ejecución. Si ambos códigos son iguales es porque la aplicación se está ejecutando en la PC correcta.

- ***RF 6. Mostrar Información referente a la Licencia.***

El sistema deberá ser capaz de acceder al archivo de licencia descodificar el contenido y mostrar por separado cada información de la licencia, ya que el sistema cliente es el que determina como quedará configurado el formulario que mostrará la información de la licencia.

- ***RF 7. Generar llave de licencia.***

El sistema, a partir de un código de activación, generará una llave de licencia para el licenciamiento de la aplicación que se desea proteger además incorporará a dicha llave información adicional, la cual será de

utilidad para la identificación de la llave generada. Esta llave de licencia tendrá la particularidad de que será única, y solo podrá ser utilizada en la PC donde coincidan las características del hardware brindadas, para la cual fue creada.

- ***RF 8. Encriptar llave de licencia.***

Para obtener una mayor seguridad la llave de licencia creada deberá ser encriptado utilizando llaves públicas y privadas. De esta forma se lograra que la aplicación posea una mayor solidez en su funcionamiento.

- ***RF 9. Guardar Archivo de Licencia.***

El sistema deberá permitir guardar el archivo de licencia, automáticamente el sistema debe sugerir una ubicación para guardar dicho archivo, pero el encargado de generar la licencia deberá poder escoger la ubicación deseada para guardar su llave de licencia.

- **RF 10. Renovar Archivo de Licencia.**

Este requisito permitirá al sistema cliente poder cambiar su llave de licencia cuando la misma haya expirado.

2.3.2. Requisitos no Funcionales.

Un requisito no funcional es un requisito de software que describe no lo que el software hará, sino cómo lo hará. Los requisitos no funcionales son difíciles de verificar/testear, y por ello son evaluados subjetivamente (González, 2007). Se puede decir que los requisitos no funcionales constituyen la forma en la que debe actuar el sistema para que funcione de manera eficaz, atendiendo aspectos tales como la disponibilidad, flexibilidad, seguridad y facilidad de uso.

- ✓ **Usabilidad**

RNF 1: El componente para la protección de software debe proveer una clase de interfaz solo con las funcionalidades de interés para el cliente.

RNF 2: El componente debe ser desarrollado en forma de biblioteca para una mejor integración con los sistemas clientes.

- ✓ **Portabilidad**

RNF 3: El componente para la protección de software debe ser ejecutado en sistemas operativos Linux.

- ✓ **Requisitos de hardware.**

RNF 4: El sistema podrá ser ejecutado en ambientes de trabajo que soporten sistemas operativos Linux. Exceptuando las versiones que ya no reciben soporte.

- ✓ **Requisitos en el diseño e implementación.**

RNF 5: El lenguaje de programación para la implementación del componente deberá ser C++.
RNF 6: Se deberá utilizar el marco de trabajo Qt en su versión 4.8.
RNF 7: Como entorno integrado de desarrollo se utilizará QtCreator en su versión 2.6.
RNF 8: Como lenguaje de modelado se utilizará UML.
RNF 9: Visual Paradigm para UML en su versión 8.0.

- ✓ **Seguridad.**

RNF 10: El componente deberá contar con una clase de interfaz capaz de brindar solo la información necesaria al sistema cliente, para que este solo tenga dominio de las funcionalidades que necesita para ejercer su protección evitando de esta manera que se conozca lo que sucede en la lógica de negocio.

RNF 11: El archivo de licencia deberá ser encriptado utilizando técnicas que aseguren su integridad y la confidencialidad de la información contenida en la licencia.

RNF 12: La biblioteca para encriptar el contenido de la llave de licencia deberá ser libqca.

- ✓ **Interfaz:**

RNF 13: Las interfaces gráficas implementadas para el sistema deben concebirse con un ambiente sencillo para el usuario.

Para obtener un resultado satisfactorio en la usabilidad de la aplicación es necesario regirse por los principios de diseño ya que estos son fundamentales para diseñar e implementar interfaces gráficas de usuario efectivas y usables. También existe un conjunto de reglas que se deben aplicar en el desarrollo de la interfaz gráfica de usuario del sistema por las que se va a guiar este trabajo de investigación para lograr la usabilidad del sistema.

Reglas de usabilidad:

- ✓ **Control y libertad del usuario:** La interfaz debe ser diseñada de tal manera que el control de la interacción con el sistema lo tenga el usuario de manera que interactúe directamente con los objetos de

la pantalla, haciéndolo más cómodo y no como un módulo más de la aplicación. Esto se consigue cuando el usuario manipula los objetos como si fueran objetos físicos.

- ✓ **Consistencia y estándares:** Una buena interfaz contribuye al aumento de la productividad si es consistente en todos los diálogos que desarrolla, basándose en el conocimiento que el usuario ha adquirido con otras aplicaciones y en la aplicación propia. Se debe mantener la consistencia en todas las aplicaciones relacionadas. Deberán implementar las mismas reglas de diseño para mantener la consistencia en toda la interacción. El usuario debe ser capaz de saber en cada momento en qué contexto está trabajando, de donde viene y a donde va. Esto se puede realizar con indicadores gráficos como iconos o colores diferentes para cada situación.
- ✓ **Correspondencia entre el sistema y el mundo real:** El sistema debe hablar el lenguaje de los usuarios, con palabras, frases y conceptos familiares para el usuario, siempre en el contexto de la aplicación. Se debe hacer que la información aparezca en un orden lógico y natural. El usuario no tiene por qué conocer los términos técnicos utilizados en el mundo informático. La aplicación debe interactuar con el usuario de forma que este perciba las palabras y frases cotidianas de la metáfora de la aplicación. La aplicación debe ser lo más parecida posible al objeto del mundo real que representa.
- ✓ **Estética y diseño minimalista:** Los diálogos no deben contener información que sea irrelevante para la tarea que está realizando el usuario. Debe ser una interfaz simple, fácil de aprender y de usar y con fácil acceso a las funcionalidades que ofrece la aplicación. (Alcala, 2007)

2.4. Diagramas de casos de uso del sistema.

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Además es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema.

Sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo. Los personajes o entidades que participarán en un caso de uso se denominan actores (Addison, 1992). La siguiente figura muestra el diagrama de casos de uso correspondiente al componente que se desea desarrollar.

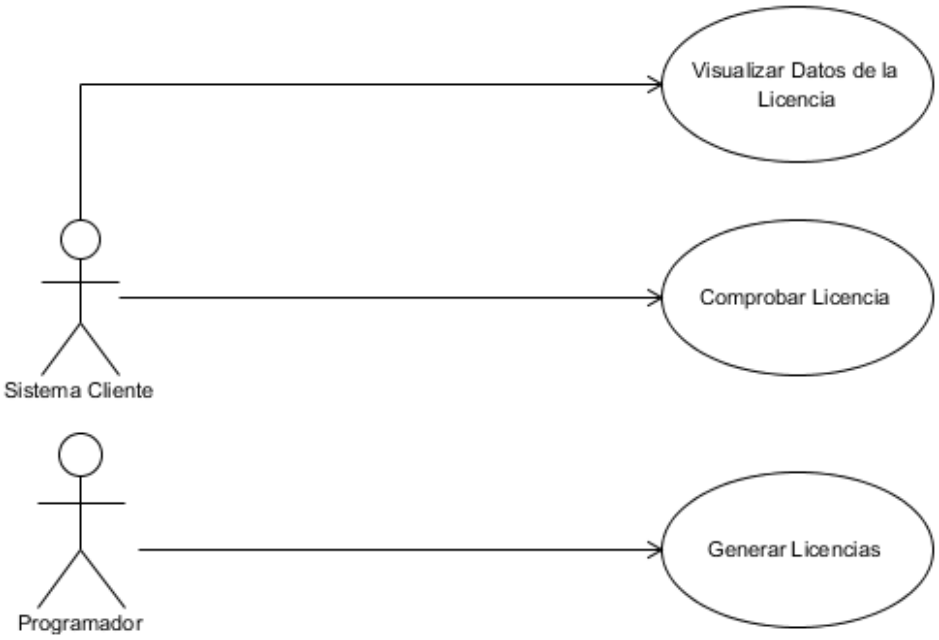

Figura 2. Diagrama de casos de uso del sistema.

2.4.1. Definición de los Actores.

Actor	Descripción
Programador	El Programador es la persona que deberá dar soporte para las licencia, es decir, si el cliente solicita una nueva licencia es este quien debe crearla con toda la información necesaria y facilitársela al cliente.
Sistema Cliente	El actor Sistema Cliente es el encargado de inicializar los casos de uso Comprobar Licencia y Mostrar Información de la licencia, estos casos de usos son ejecutados por el sistema cliente automáticamente cuando se ejecuta.

Tabla 1. Descripción de los Actores.

2.4.2. Especificación de los Casos de Uso del Sistema (CUS).

Descripción del CUS Generar Licencia.

Objetivo	Generar Licencia	
Actores	Programador	
Resumen	El caso de uso comienza cuando el Programador ejecute el Generador de Licencias, luego se muestra una interfaz, donde se proporcionarán datos al sistema para que este genere la llave de licencia. El caso de uso termina cuando la llave haya sido generada y el actor cierre la aplicación.	
Complejidad	Alta	
Prioridad	Alta	
Precondiciones	1- Se debe contar con un código de activación proporcionado por el sistema. 2- Debe estar instalada la biblioteca libqca y todas sus dependencias.	
Postcondiciones	La licencia fue generada exitosamente.	
Flujo de eventos		
Flujo básico: Generar Licencia.		
	Actor	**Sistema**
1.	El Programador ejecuta la aplicación para generar las licencias.	
2		El sistema muestra una interfaz en la cual el programador deberá proporcionar la información necesaria para que el sistema genere la llave de

		licencia, esta información estará compuesta por el nombre del producto que se desea proteger, el nombre de la persona que representa la aplicación cliente, un código de activación compuesto de 32 caracteres, la fecha de inicio de la licencia y la fecha de vencimiento de la misma.
3.	El encargado proporciona la información necesaria al sistema y presiona el botón Guardar.	
4.		El sistema verifica que no existan campos vacíos, luego verifica que el código de activación introducido tenga la longitud requerida. El sistema obtiene toda la información brindada por el programador, guarda toda la información en una variable, encripta la información utilizando el algoritmo AES[16] 1024, luego muestra una interfaz en la que permite al programador escoger el lugar donde desea guardar la llave de licencia, automáticamente el sistema le asigna un nombre al archivo de licencia y espera a que el usuario presione el botón Guardar.
5.	El programador escoge la ubicación donde va a guardar la llave de licencia y presiona el	

[16] **AES**: Esquema de cifrado por bloques adoptado como un estándar de cifrado por el gobierno de los Estados Unidos. Algoritmo más usado en criptografía simétrica.

	botón Guardar.	
6.		El sistema guarda en un archivo con extensión .key el contenido de la llave de licencia en la ubicación especificada y con el nombre asignado, además emite un mensaje indicando que la llave de licencia ha sido generada satisfactoriamente, terminando de esta forma el caso de uso.
Flujos alternos		
4a. Se detectaron campos vacíos.		
	Actor	**Sistema**
4a.	El encargado proporciona la información necesaria al sistema y presiona el botón Guardar.	
		Si el sistema detecta la existencia de algún campo vacío, notificará a través de un mensaje que existen campos vacíos y que deben ser llenados.
4b.	El encargado proporciona la información necesaria al sistema y presiona el botón Guardar.	
		Si el sistema detecta que el código de activación proporcionado por el programador no contiene la longitud requerida, emitirá un mensaje indicando tal situación y pedirá que sea

		rectificada.
5a.	El programador escoge la ubicación donde va a guardar la llave de licencia.	
		En caso de que la ubicación seleccionada por el usuario no pueda ser accedida por el sistema este notificará que no se puede guardar el archivo de licencia en dicha ubicación.
Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos no funcionales	RF 7, RF 8, RF 9	
Asuntos pendientes		

Tabla 2. Descripción del CUS Generar Llave de Licencia.

Descripción del CUS Comprobar Llave.

Objetivo	Comprobar la autenticidad de la llave de licencia.
Actores	Sistema Cliente
Resumen	El caso de uso se inicializa cuando el sistema a proteger sea ejecutado, luego el componente para la protección de software comprueba la existencia de la llave de licencia y que todos los parámetros de dicha llave estén correctos, luego si todo está en orden se ejecuta la aplicación que se está protegiendo. El caso de uso termina cuando el componente para la protección de software determine que la llave de licencia está configurada correctamente y ejecuta la aplicación protegida.

Complejidad	Alta	
Prioridad	Alta	
Precondiciones	1- Debe estar instalada la biblioteca libqca y todas sus dependencias.	
Postcondiciones	El sistema permite la ejecución de la aplicación cliente.	
Flujo de eventos		
Flujo básico: Comprobar Llave		
	Actor	**Sistema**
1	El sistema cliente ejecuta el componente para la protección de software (sistema).	
2		El sistema comprueba la existencia del archivo de licencia, luego verifica que los códigos de activación de la licencia y el obtenido en tiempo de ejecución sean iguales. Verifica que la licencia este activa mediante las fechas de inicio, la fecha actual y la de vencimiento, determina la cantidad de días restantes de la licencia, si todos las acciones ejecutadas anteriormente son correctas el sistema permite la ejecución del sistema cliente, terminando así el caso de uso.
Flujos alternos		
2.1a. No se encuentra el archivo de licencia.		
	Actor	**Sistema**
2.1.a.	El sistema cliente ejecuta el componente para la protección de software (sistema).	
		Si el archivo de licencia no se encuentra en la

		ubicación especificada y con el nombre especificado el sistema emitirá una señal indicando que el archivo de licencia no pudo ser encontrado.
2.2a. Códigos de activación incorrectos.		
2.2.a.	El sistema cliente ejecuta el componente para la protección de software (sistema).	
		Si el código de activación del archivo de licencia no se corresponde con el código de activación obtenido por el sistema en tiempo de ejecución, el sistema emitirá una señal indicando que dichos códigos no son iguales y que por tanto la licencia es incorrecta.
2.3.a.	Licencia inactiva.	
2.3.a.	El sistema cliente ejecuta el componente para la protección de software (sistema).	
		Si el sistema detecta que la llave de licencia no se encuentra dentro de su período activo o si detecta que se está ejecutando en una fecha menor a la última ejecución, el sistema emitirá una señal indicando que la llave de licencia no se encuentra activa e inhabilitará el uso del sistema cliente.
Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos	RF 1, RF 2, RF 3, RF 4 , RF 5	

Funcionales	
Asuntos pendientes	

Tabla 3. Descripción del CUS Comprobar Llave.

Descripción del CUS Visualizar Datos de la Llave de Licencia.

Objetivo	Mostrar todos los parámetros pertenecientes a la licencia.	
Actores	Sistema Cliente	
Resumen	El caso de uso se inicializa cuando el sistema cliente desea mostrar la información concerniente a la licencia. Una vez ejecutado el sistema recolecta la información del archivo de licencia, devolviendo cada parámetro por separado, terminando de esta forma el CUS.	
Complejidad	Baja	
Prioridad	Baja	
Precondiciones	1- Debe estar instalada la librería libqca2 y todas sus dependencias.	
Postcondiciones	El sistema devuelve toda la información perteneciente a la llave de licencia.	
Flujo de eventos		
Flujo básico: Comprobar Llave		
	Actor	**Sistema**
1	El sistema cliente solicita mostrar la información de la llave de licencia.	
2		El sistema accede al archivo de licencia, descodifica la información y proporciona por

		separado cada parámetro de la misma.
Flujos alternos		
2.1a. No existe el archivo de licencia.		
	Actor	**Sistema**
2.1.a.	El sistema cliente solicita mostrar la información de la llave de licencia.	
		Si el archivo de licencia no puede ser accedido el sistema emitirá una señal indicando la no existencia de dicho archivo de licencia.
2.2a. Licencia no activa.		
2.2.a.	El sistema cliente solicita mostrar la información de la llave de licencia.	
		Si el sistema no puede descodificar la información del archivo de licencia emitirá una sea indicando que no ha podido descodificar la información.

Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos Funcionales	RF6, RF10	
Asuntos pendientes		

Tabla 4. Descripción del CUS Visualizar Datos de la Llave de Licencia.

2.5. Arquitectura.

La arquitectura de software es la estructura del sistema que comprenden elementos de software, las propiedades visibles externamente de esos elementos y las relaciones entre ellos. Una arquitectura de software puede estar basada en elementos sencillos o componentes prefabricados de mayor tamaño, y se especifica de acuerdo con los diferentes tipos de sistemas. Debe ser escogida de manera que minimice los efectos de cambios futuros en el sistema. (Weitzenfeld., 2000)

La arquitectura no es el software operacional. Más bien, es la representación que capacita al ingeniero del software para:

- Analizar la efectividad del desafío para la consecución de los requisitos fijados.
- Considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil.
- Reducir los riesgos asociados a la construcción del software. (Pressman, 2010)

2.5.1. Patrones de arquitectura.

Un patrón es una solución probada que se puede aplicar con éxito a determinados tipos de problemas que aparecen repetidamente en el desarrollo de software (Lleonart, 2011). Estos permiten dar solución a un problema en un contexto determinado y reutiliza soluciones a problemas comunes. Son un esqueleto básico que cada diseñador adapta a las peculiaridades de su aplicación.

2.5.2. Arquitectura N Capas.

Esta arquitectura soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite la partición de un problema complejo en una secuencia de pasos incrementales. Admite muy naturalmente optimizaciones y refinamientos. Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas (Carlos, 2009). Para el desarrollo de la siguiente investigación se toma como variante la arquitectura en tres capas.

Variante en tres capas.

Para el desarrollo del componente para la protección de software que se desea implementar se definió una arquitectura en tres capas como una variante del patrón N-capas: la capa de presentación es la encargada

de velar por el usuario, ya que presenta al usuario la interfaz del componente, le permite la comunicación con el mismo, captura toda la información provista por el usuario, realizando un filtrado previo para comprobar la inexistencia de errores. Se caracteriza además, por tener una interfaz amigable, entendible y fácil de utilizar por el usuario, es la encargada de comunicarse directamente con la capa de negocio. La capa de negocio es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso, es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación y la capa de acceso a datos, para recibir las solicitudes y presentar los resultados. La capa de acceso a datos es la encargada de almacenar los datos del sistema, su función es almacenar y devolver datos a la capa de negocio, en este caso la información almacenada en las clases de entidades correspondientes a cada dispositivo seleccionado. A continuación se muestra una representación gráfica de la arquitectura en tres capas.

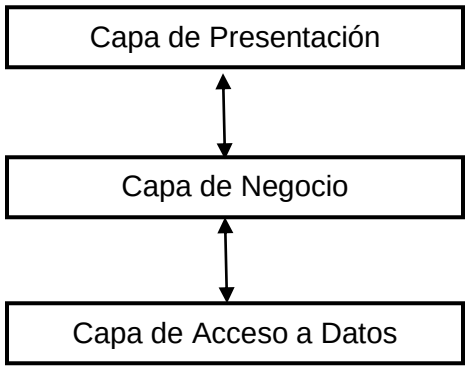

Figura 3. Representación gráfica de la variante en tres capas.

Se escogió esta variante de la arquitectura N-Capas para el desarrollo del componente, ya que con la puesta en práctica se reducen las dependencias existentes entre las clases, de esta forma las capas situadas en los niveles más bajos no tienen conocimiento de lo que ocurre en las capas superiores, permitiendo la modificación de una capa sin afectar a las otras. La comunicación sería solo entre capas adyacentes, posibilitando que el componente tenga una arquitectura escalable, al concentrar en cada capa las funcionalidades pertinentes. Esta organización permitirá una mejor organización durante el desarrollo así como brindar un mejor soporte al sistema.

2.5.3. Patrones de diseño.

Los patrones intentan codificar el conocimiento, las expresiones y los principios ya existentes: cuanto más

trillado y generalizados, tanto mejor. Estos brindan una serie de recomendaciones para organizar los distintos componentes del sistema y así tener un mejor entendimiento del problema al que se le quiere dar solución. Para la solución de esta aplicación se tuvo en cuenta diversos patrones de diseño, los cuales se presentan a continuación: (Craig, 2008)

Los patrones GRASP (General Responsibility Assignment Software Patterns o Patrones Generales de Software para Asignar Responsabilidades) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. (Alcolea, 2012)

Los patrones GoF (Gang of Four o Grupo de Cuatro) son soluciones técnicas basadas en Programación Orientada a Objetos (POO). Se clasifican en patrones estructurales, creacionales y de comportamiento. (Expósito, 2011)

Patrones GRASP utilizados en el desarrollo del componente.

- ✓ **Experto:** Este patrón, está diseñado para que la responsabilidad de realizar una labor sea de la clase que tiene o puede tener los atributos involucrados. Este patrón está presente en las clases Crypto_Tools y Crypto_Cypher, las cuales solo contienen la información necesaria para realizar las operaciones de cifrar y descifrar.
- ✓ **Creador:** Guía la asignación de responsabilidades relacionadas con la creación de objetos. La intención básica del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Este patrón se pone de manifiesto en la clase SProtection cuando se crean instancias de las clases entidades, a la hora de crear los objetos con la información pertinente, es decir, se crean los objetos de cada clase y se les pasa a través de su constructor toda la información que se necesita.
- ✓ **Controlador:** Está diseñado para asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades. El patrón controlador se pone de manifiesto en la clase SProtection la cual es la encargada de controlar el flujo de eventos del sistema.
- ✓ **Alta cohesión:** Propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo. Este patrón se pone de manifiesto durante todo el desarrollo del componente, ya que cada clase se encarga solamente de hacer lo que le corresponde, evitando la dependencia entre las clases.

- ✓ **Bajo Acoplamiento:** Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas no se puede extraer software de forma independiente y reutilizable. Este patrón está presente siempre que se asegure la alta cohesión.

Patrones GoF utilizados en el desarrollo del componente.

Fachada: Define una clase con una interfaz común a un grupo de componentes o a un conjunto heterogéneo de interfaces, a esa clase le damos el nombre de Fachada. Los elementos heterogéneos pueden ser las clases de un paquete, un conjunto de funciones, un esquema o un subsistema (local o remoto) (Craig, 2008). Este patrón es evidenciado en la clase SProtection, la cual solo contiene y presenta al usuario final las funcionalidades que son de interés para él.

Observador: Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él. Al utilizar el mecanismo de señales y slots que implementa el framework de desarrollo escogido, se pone de manifiesto el patrón Observador, este patrón es utilizado cuando se va a cifrar la licencia, ya que se observa constantemente a que se presione un botón.

2.6. Modelo del análisis.

La metodología RUP posibilita decidir para el flujo de trabajo análisis y diseño las actividades que deben llevarse a cabo. Indicando las partes que pueden introducirse de forma relativa independiente del resto. El modelo de análisis es un artefacto generado en este flujo de trabajo, útil para comprender mejor los requisitos antes de tomar decisiones de diseño y proporcionar una visión general conceptual del sistema, pero definido por RUP como un modelo opcional que puede ser sustituido por el modelo de diseño. En la presente investigación se presentan solo las actividades correspondientes al diseño, excluyendo el análisis. Esto facilita disminuir el tiempo de desarrollo puesto que el cliente necesita una solución a corto plazo.

2.7. Modelo del diseño.

En el diseño se modela el sistema para que soporte todos los requisitos, es importante el resultado del análisis ya que el mismo proporciona una descripción detallada de los requisitos y da una estructura del sistema. El mismo es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas en el entorno de la implementación, tienen impacto en el sistema a considerar. Además sirve de abstracción

de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación.

2.7.1. Diagrama de Clases del Diseño.

El diagrama de clase del diseño describe gráficamente las especificaciones de las clases del software y de las interfaces en una aplicación. Conteniendo información como: clases, asociaciones y atributos, interfaces, con sus operaciones y constructores, métodos, tipos de atributos y dependencias. La siguiente figura muestra el diagrama de clases del diseño para el Componente para la Protección de Software.

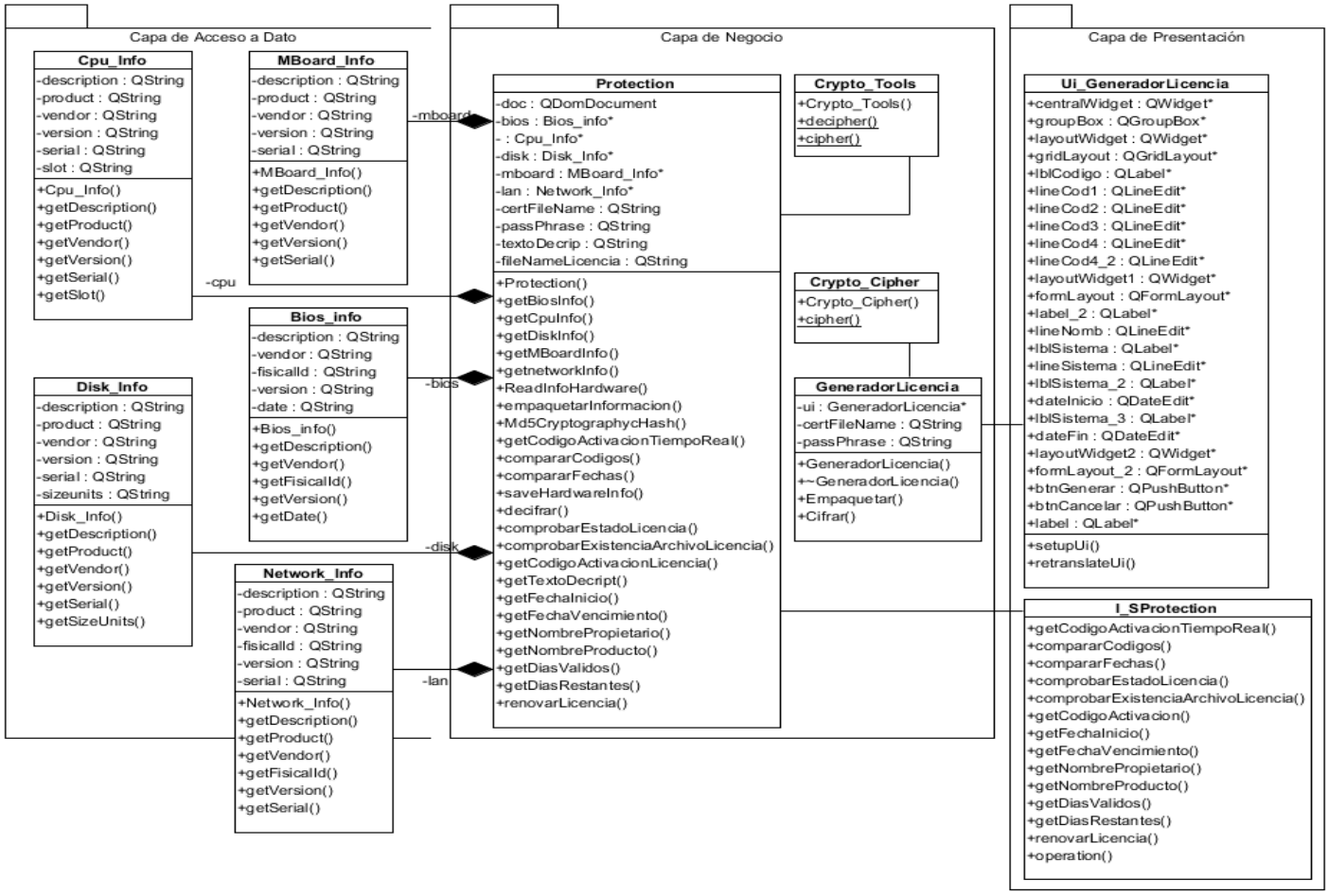

Figura 4. Diagrama de clases del diseño del componente para la protección de software.

2.7.2. Diagramas de Secuencia.

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso, contiene detalles de implementación del escenario, incluyendo

los objetos y clases que se usan para implementar el escenario y mensajes intercambiados entre los objetos (Craig, 2008)

Típicamente se examina la descripción de un caso de uso para determinar qué objetos son necesarios para la implementación del escenario. Si se dispone de la descripción de cada caso de uso como una secuencia de varios pasos, entonces se puede "caminar sobre" esos pasos para descubrir qué objetos son necesarios para que se puedan seguir los pasos. Un diagrama de secuencia muestra los objetos que intervienen en el escenario con líneas discontinuas verticales, y los mensajes pasados entre los objetos como flechas horizontales. Las siguientes figuras muestran los diagramas de secuencias correspondientes a los casos de uso Generar Licencias, Comprobar Licencia y Renovar Licencia consecutivamente.

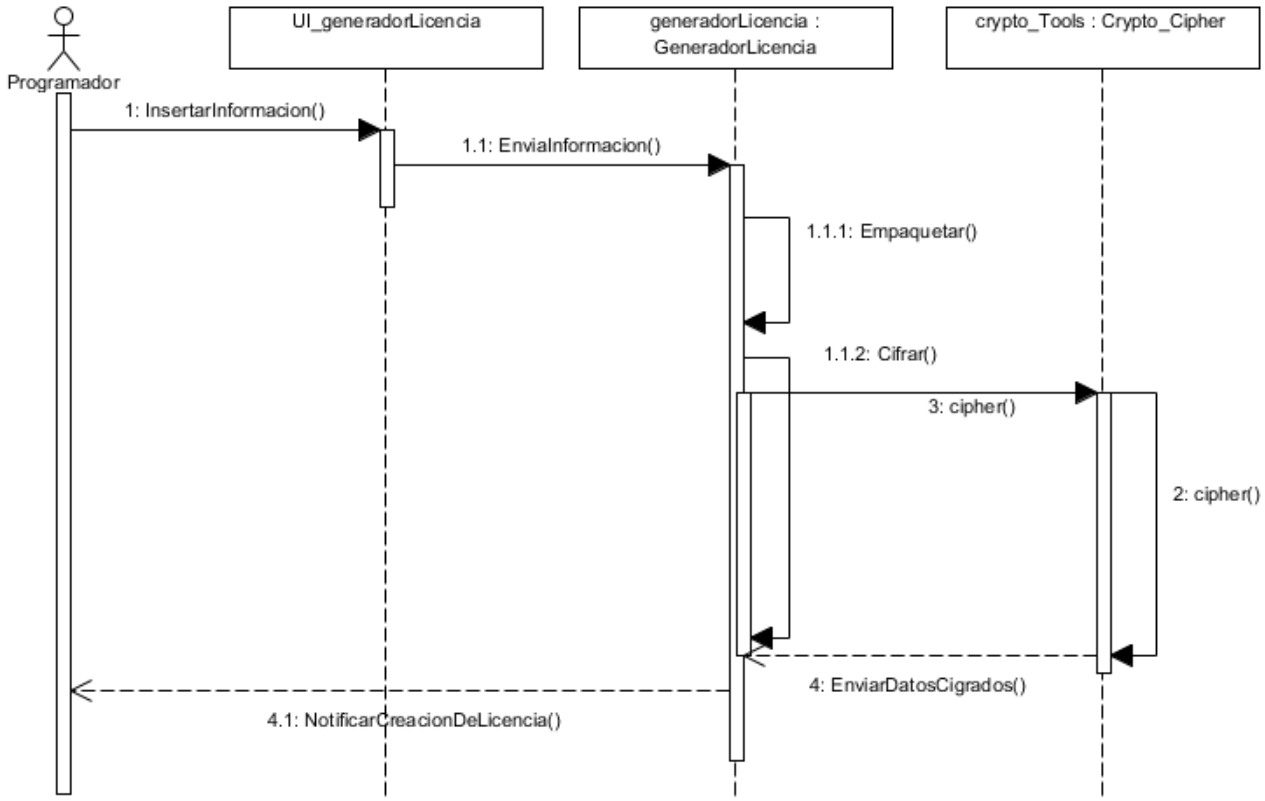

Figura 5. Diagrama de Secuencia CUS Generar Licencia

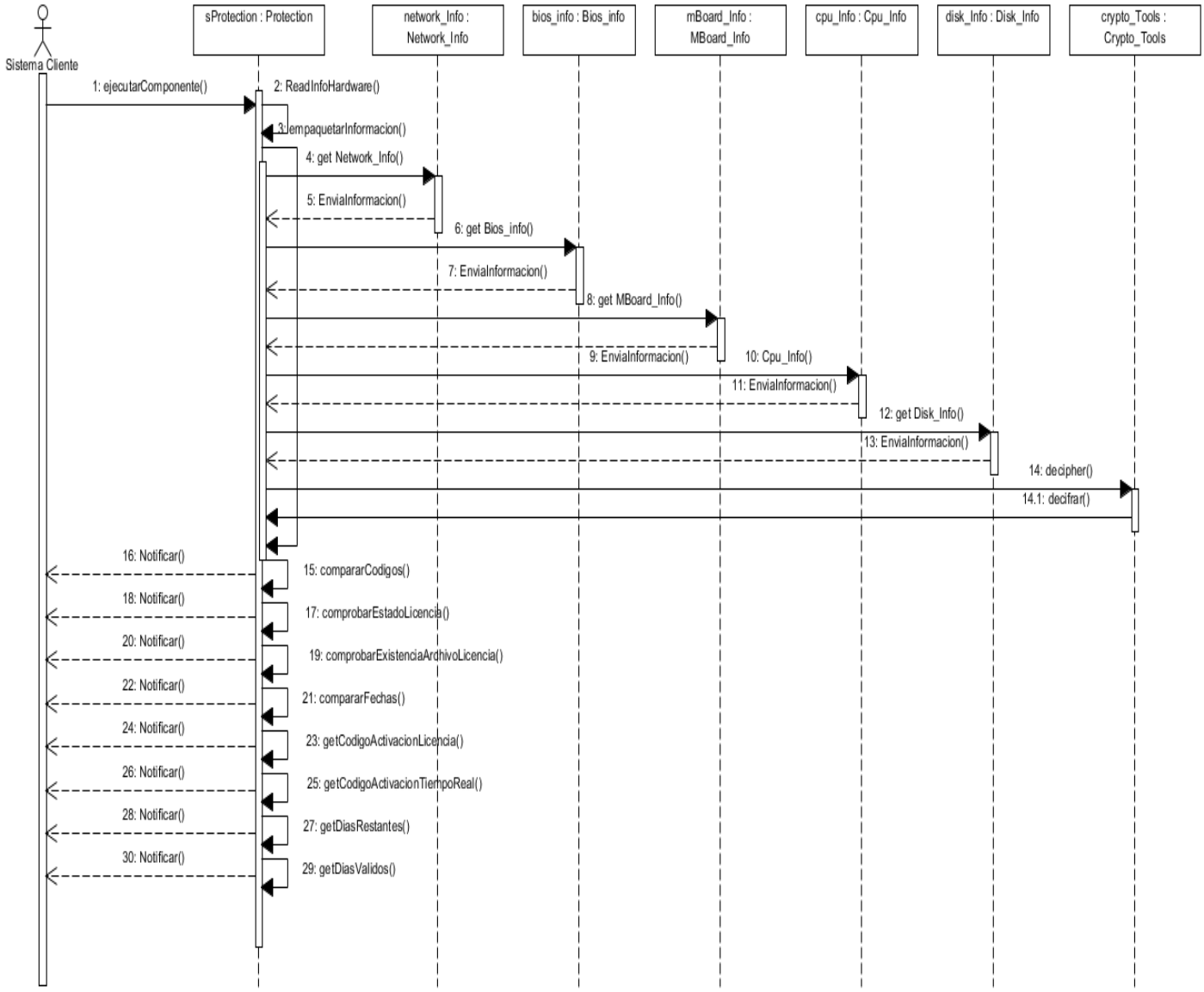

Figura 6. Diagrama de secuencia CUS Comprobar Licencia.

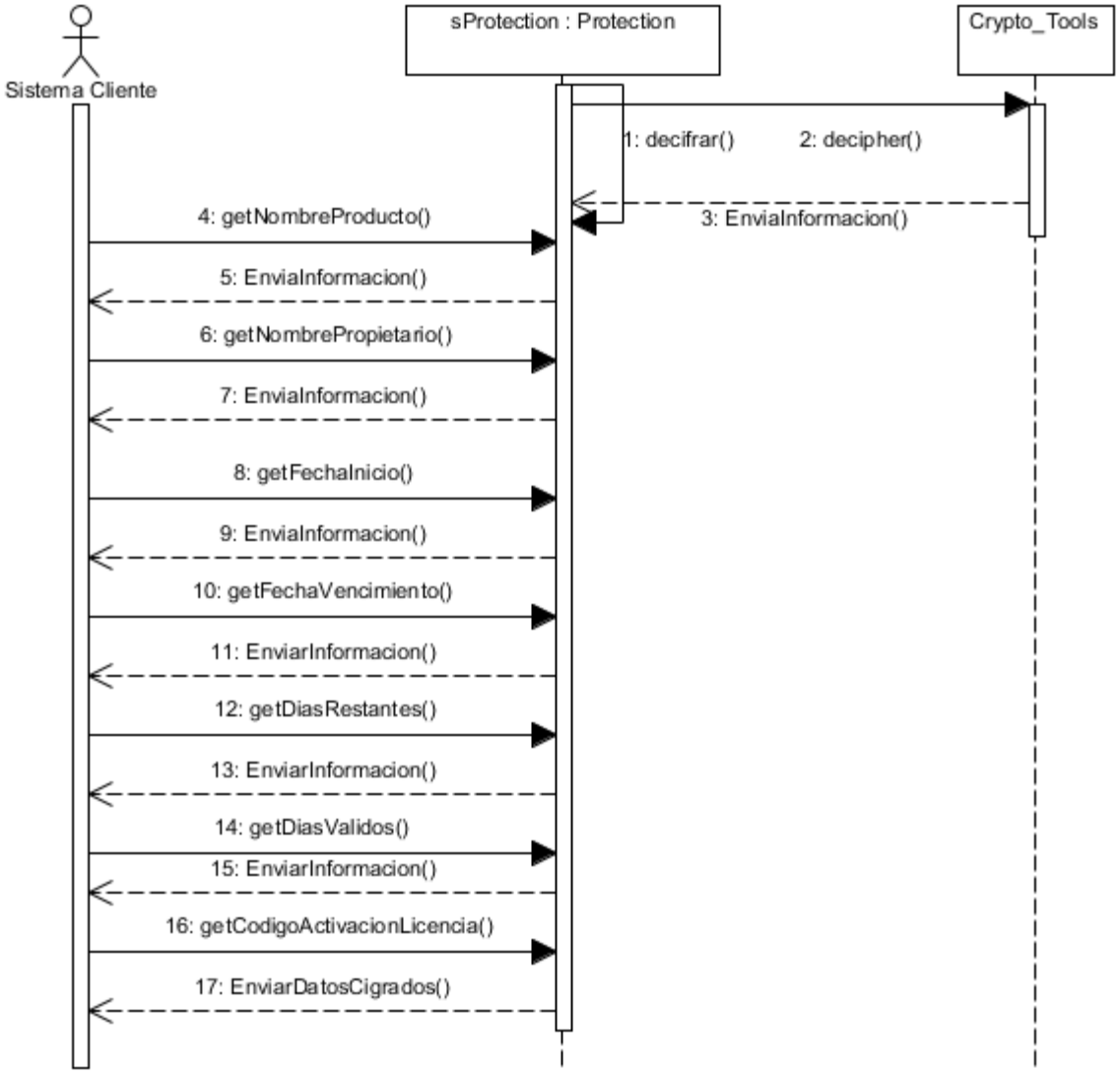

Figura 7. Diagrama de secuencia CUS Renovar Licencias

2.8. Conclusiones.

En este capítulo se trataron temas referentes al análisis y diseño del componente para la protección de software lo que permitió arribar a las siguientes conclusiones:

- ✓ La modelación de las clases del dominio permitió establecer un punto de partida para lograr un correcto diseño del sistema, permitiendo además la obtención de un mejor razonamiento del problema a resolver mediante la presentación de un diagrama y los principales conceptos que intervienen en dominio
- ✓ Se estableció la línea base de la arquitectura que tendrá el componente a desarrollar empleándose el patrón arquitectónico en tres capas como una especificación de la arquitectura N-capas, facilitando la reutilización de las capas, la estandarización y la contención de cambios, proporcionando con esto que se logre garantizar el mejor desempeño, robustez, portabilidad, flexibilidad y escalabilidad de la aplicación.
- ✓ Se identificaron los patrones de diseño GRASP y GoF utilizados en el desarrollo de la aplicación, permitiendo facilitar la asignación de responsabilidades logrando un diseño de software que sirva de apoyo a la implementación del sistema.
- ✓ Con la realización del diagrama de clases del diseño se obtuvo una visión más exacta del sistema en términos de implementación, lo que ayudará al programador a tener una visión más exacta de lo que se desea implementar.
- ✓ La representación de los diagramas de casos de uso y sus respectivas descripciones, así como la representación de los diagramas de secuencia, permitieron al desarrollador tener una vista más clara de lo que se pretende hacer.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

3.1. Introducción.

En este capítulo se describe la implementación y las pruebas que se le realizarán al sistema, después de haber realizado el análisis y diseño del mismo en el capítulo anterior. La implementación comienza con el resultado del diseño y se implementa en términos de componentes, es decir, ficheros de código fuente, ficheros de código binario, ejecutables y similares. El flujo de trabajo de implementación detalla cómo los elementos del modelo del diseño se implementan en términos de componentes y representa cómo se organizan en el modelo de despliegue, describiendo los componentes a construir, su organización y dependencias entre los nodos físicos en la que funcionará la aplicación.

3.2. Modelo de Implementación.

El modelo de implementación describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el entorno de implementación y en el lenguaje de programación utilizado, y cómo dependen los componentes unos de otros. Identifica los componentes físicos de la implementación para que puedan comprenderse y gestionarse mejor (Jacobson, 2000). El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. A continuación se muestra el diagrama de implementación perteneciente al Componente para la Protección de Software.

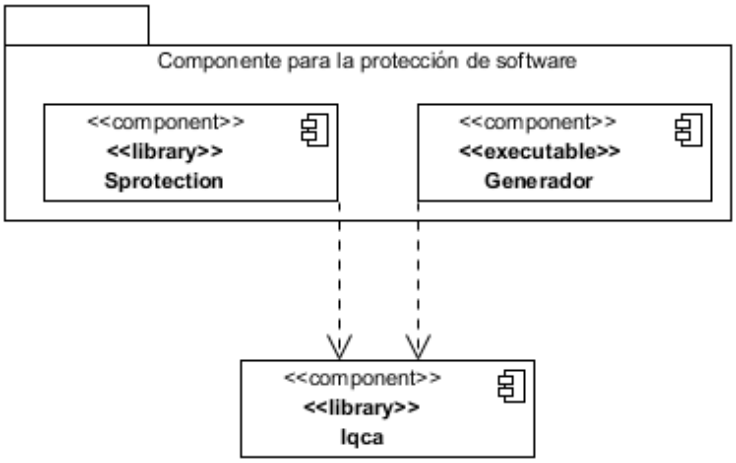

Figura 8. Modelo de Implementación.

3.2.1. Diagramas de Componentes.

Un componente es el empaquetamiento físico de los elementos de un modelo, como son las clases en un modelo de diseño. Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software, sean estos componentes de código fuente, binarios o ejecutables. Algunos estereotipos de componentes son los siguientes (Jacobson, 2000):

- <<executable>> programa que puede ser ejecutado en un nodo.
- <<file>> fichero que contiene código fuente o datos.
- <<library>> biblioteca, estática o dinámica
- <<table>> tabla de base de datos

A continuación se representa el diagrama de componentes de código fuente, del Componente para la Protección de Software.

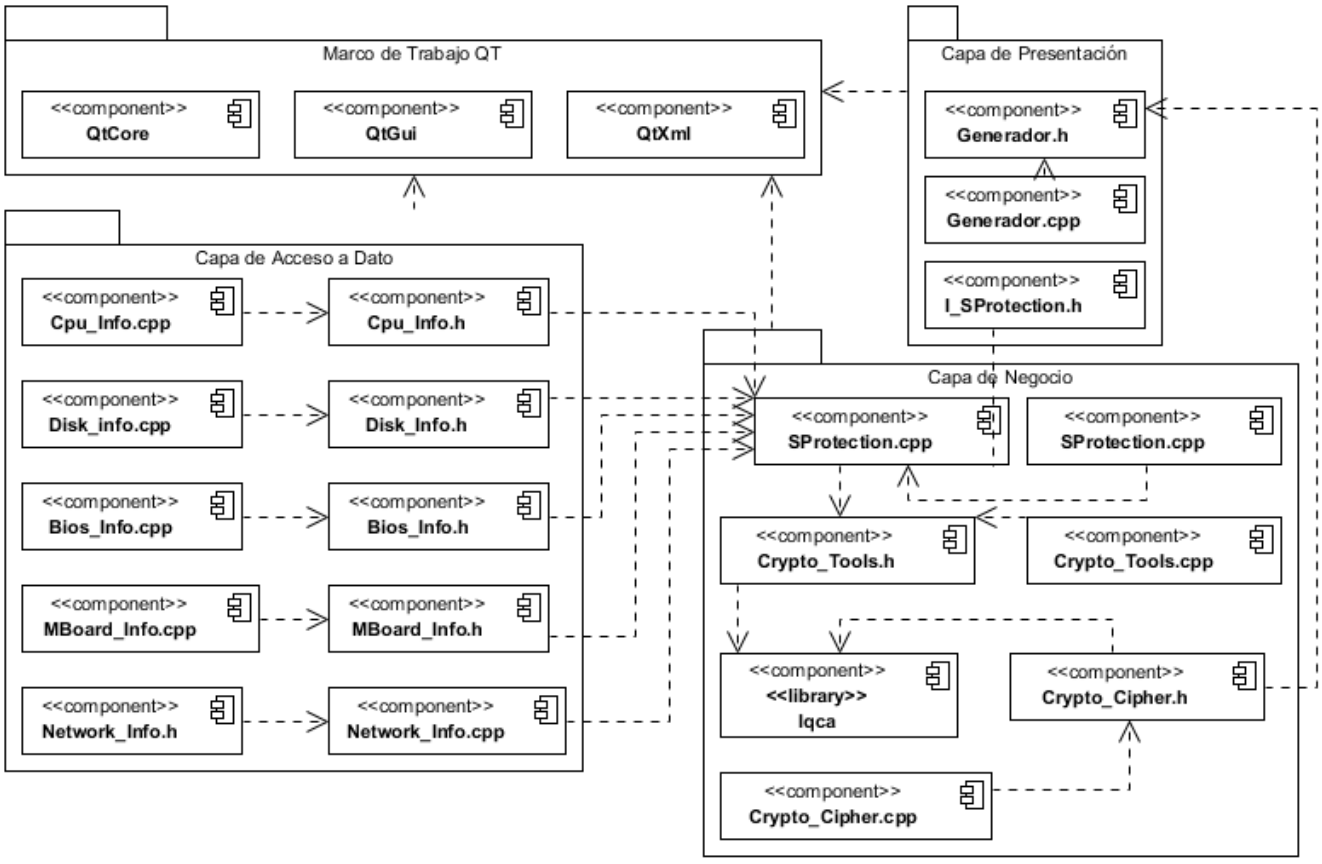

Figura 9. Diagrama de componentes de código fuente.

3.3. Modelo de Despliegue.

El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo, cada nodo representa un recurso de cómputo, normalmente un procesador o un dispositivo hardware similar. Los nodos poseen relaciones que representan medios de comunicación entre ellos, la funcionalidad de un nodo se define por los componentes que se distribuyen en ese nodo. (Salamanca, 2009)

Los nodos son los elementos donde se ejecutan los componentes, representan el despliegue físico de los componentes. Los componentes son los elementos que participan en la ejecución del sistema, representan el empaquetamiento físico de los elementos lógicos (Xavier, 2010).

En esta vista se realiza una representación gráfica de los nodos físicos en los que estará desplegado el sistema propuesto y la comunicación entre ellos. El siguiente diagrama de despliegue muestra la distribución física que tendrá el sistema una vez puesto en práctica.

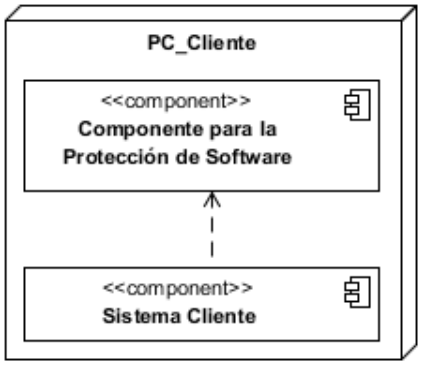

Figura 10. Diagrama de despliegue.

3.4. Prueba del Software.

Cuando se habla de las pruebas de software se puede definir de la siguiente manera: Proceso realizado concurrentemente a través de las diferentes etapas de desarrollo de software, cuyo objetivo es apoyar la disminución del riesgo de aparición de fallas y faltas en operación**.** (Cardenas, 2009)

Las pruebas son una actividad en la cual un sistema o componente, es ejecutado bajo condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la

codificación. El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Además, esta etapa implica:

- Verificar la interacción de componentes.
- Verificar que todos los requisitos se han implementado correctamente.
- Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
- Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

3.4.1. Técnicas de Prueba.

Las técnicas de pruebas permitirán examinar los detalles procedimentales del código y la lógica del programa así como la interfaz y su funcionalidad. Por medio de las diferentes técnicas se brindan criterios variados para generar casos de pruebas que provoquen fallos en los programas. Es por ello que es de vital importancia conocer dichas técnicas.

- ✓ Pruebas de caja negra: Realizar pruebas de forma que se compruebe que cada función es operativa.
- ✓ Pruebas de caja blanca: Desarrollar pruebas de forma que se asegure que la operación interna se ajusta a las especificaciones, y que todos los componentes internos se han probado de forma adecuada.

Pruebas de Caja Negra.

Un caso de prueba que especifica cómo probar un caso de uso o un escenario específico de un caso de uso. Un caso de prueba de este tipo incluye la verificación del resultado de la interacción entre el actor y el sistema, que se satisfacen las precondiciones y pos condiciones especificadas por el caso de uso y que se sigue la secuencia de acciones específicas por el caso de uso. Un caso de prueba basado en un caso de uso especifica típicamente una prueba del sistema como caja negra, es decir, una prueba del comportamiento observable externamente del sistema.

Estas pruebas permiten encontrar:

- Funciones que estén incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.

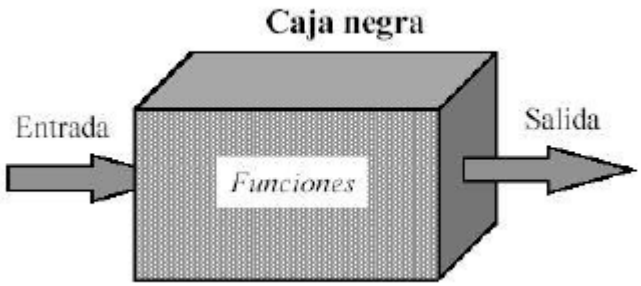

Figura 11. Representación gráfica del proceso de pruebas de Caja Negra.

3.5. Diseño de Casos de Pruebas

Los casos de prueba especifican como probar el sistema, incluyendo la entrada o resultados con los que se harán las pruebas y las condiciones en las que se van a probar. Un caso de prueba puede derivarse y por tanto puede seguir la traza de un caso de uso.

Al Componente para la Protección de software se le realizó el método de pruebas de caja negra, las cuales se centran en verificar que el sistema cumple con los requisitos funcionales especificados en el análisis. Con la realización de las pruebas de caja negra se puede encontrar errores funcionales que puede presentar el software y no han sido detectados por los desarrolladores. Las siguientes tablas muestran los casos de prueba pertenecientes a la tercera iteración de las pruebas aplicadas al sistema.

Tabla 5. Caso de Prueba CUS Generar Licencia.

Sesión	Escenario	Acción Realizada	Reacción del Sistema	Resultado
	EC 1.1. Licencia generada	El administrador introduce los datos	El sistema verifica que no existan campos vacíos,	Satisfactorio

SC 1. Generar Licencia.	satisfactoriamente.	para generar la llave de licencia y presiona el botón Guardar.	luego verifica que el código de activación y los campos restantes tenga el formato correcto y genera el archivo de licencia en la ubicación especificada por el programador.	
	SC 1.2. Se detectaron campos vacíos.	El administrador introduce los datos para generar la llave de licencia y presiona el botón Guardar.	Si el sistema detecta que existe algún campo vacío muestra el siguiente mensaje: "*No pueden existir campos vacíos, por favor rectifique."* Además brinda la opción de volver a llenar dichos campos.	Satisfactorio.
	SC 1.3. Numero de Activación Incorrecto.	El administrador introduce los datos para generar la llave de licencia y presiona el botón Guardar.	Si el sistema detecta que el código de activación no tiene la longitud especificada y detecta la existencia de caracteres no válidos, emitirá el siguiente mensaje: *"El código de activación no es correcto. Por favor rectifique que no existan caracteres extraños y que tenga una longitud de 32 caracteres"*, indicando que el código de activación	Satisfactorio.

			está incorrecto y volverá a brindar la opción de rectificarlo.	

Tabla 6. Caso de Prueba CUS Comprobar Licencia.

Sesión	Escenario	Acción Realizada	Reacción del Sistema	Resultado
SC 1. Comprobar Licencia.	EC 1.1. Llave de licencia correcta.	El sistema cliente ejecuta el componente para la protección de software (sistema).	El sistema comprueba la existencia del archivo de licencia, verifica que los códigos de activación de la licencia y el obtenido en tiempo de ejecución sean iguales. Verifica que la licencia está activa mediante las fechas de inicio, la fecha actual y la de vencimiento, determina la cantidad de días restantes de la licencia, permitiendo la interacción con el sistema cliente.	Satisfactorio
	SC 1.2. No se encuentra el archivo de licencia.	El sistema cliente ejecuta el componente para la protección de software (sistema).	Si el archivo de licencia no se encuentra en la ubicación especificada y con el nombre especificado el sistema	Satisfactorio.

			emitirá una señal indicando que el archivo de licencia no se encuentra e inhabilitará al sistema cliente.	
	SC 1.3. Códigos de activación incorrectos.	El sistema cliente ejecuta el componente para la protección de software (sistema).	Si el código de activación del archivo de licencia no se corresponde con el código de activación obtenido por el sistema en tiempo de ejecución, el sistema emitirá una señal indicando que dichos códigos no son iguales y que por tanto la licencia es incorrecta inhabilitando el uso del sistema cliente.	.
	SC 1.4. Licencia inactiva.		Si el sistema detecta que la llave de licencia no se encuentra dentro de su período activo o si detecta que se está ejecutando en una fecha menor a la última ejecución, el sistema emitirá una señal indicando que la llave de	

			licencia no se encuentra activa e inhabilitará el uso del sistema cliente.	

Tabla 7. Caso de Prueba CUS Visualizar datos de la Licencia.

Sesión	Escenario	Acción Realizada	Reacción del Sistema	Resultado
SC 1. Visualizar datos de la Licencia.	EC 1.1. Visualizar Datos de la Licencia Correctamente.	El sistema cliente solicita mostrar información de la licencia.	El sistema accede al archivo de licencia descodifica la información contenida en el archivo y devuelve la información al sistema cliente.	Satisfactorio.
	SC 1.2. Licencia no Activa.	El sistema cliente solicita mostrar información de la licencia.	El sistema descodifica el archivo de licencia y comprueba que dicha licencia este activa, en caso de no estarlo, el sistema notifica al sistema cliente mostrando la información de dicha licencia.	Satisfactorio.
	SC 1.3. Licencia no encontrada.	El sistema cliente solicita mostrar información de la licencia.	Si la licencia no puede ser accedida el sistema notificará al sistema cliente y no mostrara información referente a la licencia.	Satisfactorio.

3.5.1. Resultados de las Pruebas de Caja Negra.

Durante la fase de prueba se realizaron tres iteraciones al Componente para la Protección de software. En la primera iteración se detectó que la herramienta para generar las licencias no validaba correctamente los campos y permitía la entrada de caracteres no alfanuméricos en el campo de texto perteneciente al código de activación, además permitía la entrada de valores numéricos y caracteres extraños en el campo de texto perteneciente al nombre del propietario de la licencia. Estas inconformidades fueron corregidas y luego se volvió a realizar otra iteración de las pruebas para el caso de uso Generar Licencia, arrojando resultados satisfactorios.

Durante la segunda iteración se detectó que el algoritmo para comprobar si la licencia está activa o no, no estaba funcionando correctamente, ya que presentaba problemas a la hora de extraer las fechas del archivo de licencia, lo cual hacía que la llave de licencia siempre estuviera inactiva cuando hacía las comparaciones con la fecha actual, el problema radicaba en que el formato en el que se guardaban las fechas a la hora de generar las licencias, no se correspondían con el formato cuando se convertían de cadenas de texto a formato de fecha, esta inconformidad fue corregida y durante la tercera iteración de las pruebas los resultados para estas inconformidades fueron satisfactorios. La siguiente gráfica muestra una representación gráfica de los resultados obtenidos durante las pruebas aplicadas al sistema.

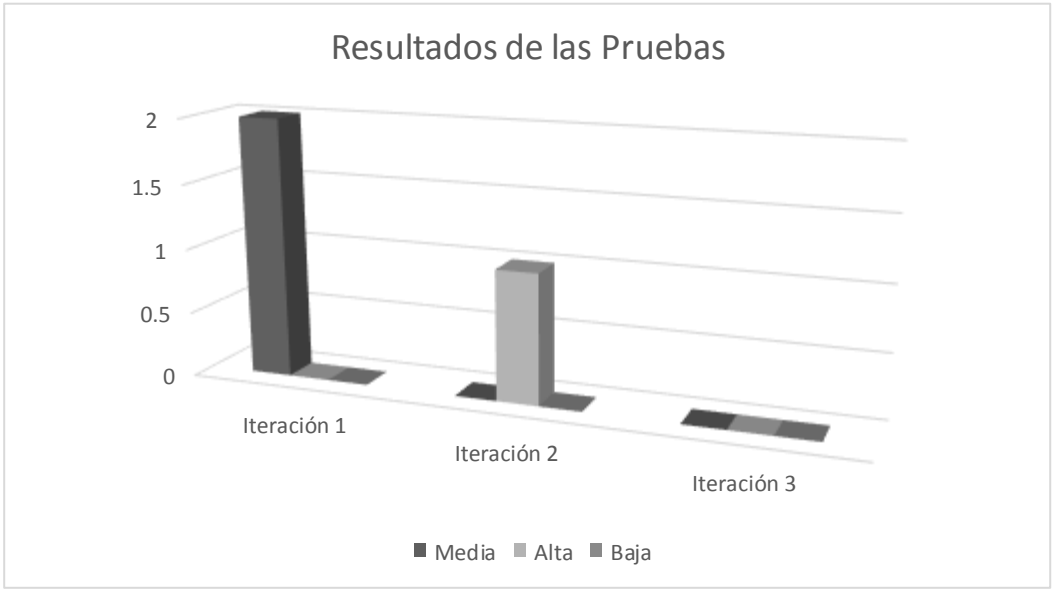

Figura 12. Resultados de las pruebas de caja negra.

3.6. Conclusiones.

En este capítulo se realizó una descripción de la implementación y las pruebas realizadas al sistema. En el mismo se generaron los artefactos necesarios para la implementación y las pruebas del sistema, por lo que se puede llegar a las siguientes conclusiones:

- ✓ La realización de los diagramas de componentes permitió comprender la interacción entre los componentes que conforman el sistema y mostrar las dependencias que existen entre los mismos.
- ✓ Mediante las pruebas de caja negra seleccionadas para validar el cumplimiento de los requisitos funcionales establecidos en el análisis del sistema, se lograron encontrar problemas funcionales presentes en el sistema.
- ✓ Con la realización de las pruebas de caja negra y sus resultados satisfactorios en la última iteración de prueba se puede concluir que el software no presenta ningún error funcional, por lo que su etapa de desarrollo culmino exitosamente.

CONCLUSIONES GENERALES

Con la realización de la presente investigación se adquirieron y pusieron en práctica los conocimientos necesarios para el desarrollo del Componente para la Protección de Software, de esta forma se le dio cumplimiento al objetivo general así como a las tareas de la investigación, arribando a las siguientes conclusiones:

- ✓ El desarrollo de la investigación científica dio lugar a la creación de las bases para el desarrollo de la presente investigación.
- ✓ El estudio de los conceptos asociados al dominio del problema ayudó en gran medida a la comprensión de los principales términos que se manejarían durante la investigación.
- ✓ La realización de un estudio a las soluciones similares existentes ayudó a determinar las principales características positivas y negativas con que cuentan los sistemas actuales de protección, tomando como premisas para el desarrollo del componente las características positivas de estos sistemas.
- ✓ Se realizó un estudio a las diferentes técnicas de protección existentes y se concluyó que la más adecuada para el desarrollo del componente es la técnica de personalización del hardware, siendo esta una de las más eficientes basadas en software.
- ✓ El análisis de las tecnologías y herramientas de desarrollo de software determinó la selección de las mismas para el desarrollo del sistema, por las características que presentaban además de permitir una mayor integración con los sistemas clientes como por ejemplo Suria 2.0.
- ✓ La descripción y el diseño del Componente para la Protección de Software, permitió crear las bases para la implementación de dicho sistema, evaluando y restringiendo cada posibilidad de error, para lograr un sistema completamente funcional.
- ✓ Las pruebas realizadas al Componente para la Protección de Software ayudaron a corregir inconformidades que no se tuvieron en cuenta durante la implementación, las mismas ayudaron a refinar el producto, liberándolo de cualquier inconformidad.

RECOMENDACIONES

Durante el desarrollo del presente trabajo han surgido ideas que podrían implementarse en una versión futura, de forma que se lograría incrementar las funcionalidades del sistema ampliando así su funcionamiento, para lo cual se recomienda:

- ✓ Implementarle al sistema una funcionalidad que sea capaz de proteger aplicaciones utilizando dispositivos externos como USB.
- ✓ Implementarle al sistema una funcionalidad que sea capaz de proteger ejecutables ya compilados.

BIBLIOGRAFÍAS Y REFERENCIAS BIBLIOGRÁFICAS.

Addison, Wesley Longman, Upper Saddle River. 1992. *Un acercamiento a través de los casos de uso.* Nueva Jersy : s.n., 1992.

Alcala, Mario Lorenzo. 2007. *Medida de la usabilidad en aplicaciones de escritorio.* 2007.

Alcolea, Walfrido Lora. 2012. *"Desarrollo de un servidor de grabación multiplataforma para el sistema de video vigilancia Suria Visión.".* La Habana, Cuba : UCI, 2012.

ALEGSA. 2011. [En línea] 2011. http://www.alegsa.com.ar/Dic/biblioteca..

Altamirano, Alfonso Valdez. 2006. *Comparativo de Entornos de Desarrollo Integrados.* 2006.

Bosch, Serge Demeyer Jan. 1998. *Object-Oriented Technologies.* 1998.

Cardenas, M. C. 2009. *Pruebas de Software. Obtenido del Documentos de investigacion.* 2009.

Carlos, Reynoso y Nicolás,Kicillof. 2009. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2009.

CodeBox. 2011. CodeBox. *CodeBox.* [En línea] 2011. [Citado el: 15 de Noviembre de 2012.] http://www.codebox.es/.

Craig, Larman. 2008. *UML y PATRONES. Introducción al análisis y diseño orientado a objeto.* 2008.

David Lie, C.T, Dan Bonneh, john Mitchell, Mark Horowitz. 2007. *Architectural Support for copy and Tamper Resistant Software.* 2007.

deficinion.org. 2012. Definición de Lenguaje de Programación. *deficinion.org.* [En línea] 2012. [Citado el: 15 de Noviembre de 2012.] http://www.definicion.org/lenguaje-de-programacion.

definicion.de. 2008. definicion.de. *definicion.de.* [En línea] 2008. [Citado el: 6 de Noviembre de 2012.] http://definicion.de/seguridad/.

Ecured. 2011. Ecured. *Ecured.* [En línea] 2011. [Citado el: 6 de Noviembre de 2012.] http://www.ecured.cu/index.php/Software.

Expósito, Yuliet Hernández. 2011. *"Desarrollo de un video sensor para la detección de objetos abandonados.".* La Habana, Cuba : UCI, 2011. 5.

Fuentes, Lidia y Antonio Vallecillo. 2010. *Una Introducción a los Perfiles UML.* 2010.

González, Oscar Diez. 2007. *Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software.* 2007.

HardKey. 2009. HardKey. *HardKey.* [En línea] 2009. [Citado el: 11 de Noviembre de 2012.] http://www.hardkey.net/.

Huerta, Antonio Villalón. 2000. *Seguridad en Unix y Redes.* 2000.

IonWorx. 2010. IonWorx. *IonWorx.* [En línea] 2010. [Citado el: 11 de Noviembre de 2012.] http://www.ionworx.com/serialshield.html.

Jacobson, Ivar, Booch, Grady and Rumbaugh, James. 2000. *El proceso unificado de desarrollo de software.* Madris : Pearson Educación, 2000. 84-78-29-036-2.

Katrib, Miguel. 2008. *Programación Orientada a Objetos en CPP.* 2008.

Lira, Jose Luis. 2011. ITESCAM. *ITESCAM.* [En línea] 2011. [Citado el: 5 de Noviembre de 2012.] www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r83598.DOCX.

Lleonart, Martín Eva and García-Menacho, Asunción. 2011. *Patrones.* Valencia : Universidad Politecnica de Valencia, 2011.

Loja. 2008. [En línea] 2008. [Citado el: 20 de Enero de 2012.] http://www.utpl.edu.ec/eva/descargas/material/175/G18401.8.pdf..

Matas, Sergi Arbona y David. 2006. *PC World Digital.* 2006.

Mora, Sergio Lujan. GPLSI. [En línea] [Citado el: 15 de Noviembre de 2012.] http://gplsi.dlsi.ua.es/~slujan/materiales/cpp-muestra.pdf..

Nokia. 2010. QT Developers Days. *QT Developers Days.* [En línea] 2010. [Citado el: 16 de Noviembre de 2012.] http://qt.nokia.com/products/developer-tools..

OMG, Object Management Group. 2007. spemarti.googlecode.com. *spemarti.googlecode.com.* [En línea] 2007. [Citado el: 5 de Noviembre de 2012.] spemarti.googlecode.com/files/UMLSuper.pdf.

Oviedo. 2009. Sitio Web de la E.U. de Ingeniería Técnica Informática de oviedo. *Sitio Web de la E.U. de Ingeniería Técnica Informática de oviedo.* [En línea] 2009. [Citado el: 16 de Noviembre de 2012.] http://petra.euitio.uniovi.es/~i1667065/HD/documentos/Entornos%20de%20Desarrollo%20Integrado.pdf..

Paradigm, Visual. 2010. Visual Paradigm. *Visual Paradigm.* [En línea] 2010. http://www.visual-paradigm.com/.

Pimentel, ErnestoAntonio Maña, Javier López, J.M. Troya. 2010. *Un Esquema Eficiente de Protección de Software Basado en Tarjetas Inteligentes.* Málaga : Universitaria, 2010. 29071.

Pressman, Roger S. 2010. *Ingenieria de Software. Un enfoque Practico.* 2010.

SafeNet. 2006. SafeNet: The Data Protection Company. *SafeNet: The Data Protection Company.* [En línea] Noviembre de 2006. [Citado el: 6 de Noviembre de 2012.] http://www.safenet-inc.com/software-monetization/sentinel-software-licensing-products/?aldn=true.

Salamanca, Universidad. Departamento de Informatica y Automatica. 2009. Modelo de Despliegue. [En línea] 2009. http://es.scribd.com/doc/49708406/19/Modelo-de-despliegue.

Sean W. Smith, S.W. 2007. *Building a High-Performance, Programmable Secure Coprocessor.* 2007.

SOFPRO, Lab Software Protection. 2011. PC Guard Software. *PC Guard Software.* [En línea] 2011. http://www.sofpro.com/pc.guard.htm.

Software Licence Management with Smart Cards. **Tuomas, Aura, D.G. 2005.** Mayo de 2005.

Techerald. 2012. Techerald. *Techerald.* [En línea] 2012. [Citado el: 15 de Noviembre de 2012.] http://techerald.com/.

Torres, Fernando. 2003. *Integración del PMBOOK al RUP para proyectos de Desarrollo de Software.* 2003.

Valencia, Universidad Politecnica de. 2006. *Introducción a RUP.* 2006.

Weitzenfeld., Alfredo. 2000. *Ingenieria de software orientada a objetos con UML. Java e Internet.* 2000.

Word Reference. 2010. *Diccionario de la lengua española.* s.l. : Espasa-Calpe, 2010.

Xavier, Segura Ferre Grau y Maria Isabel Sanchez. 2010. [En línea] 2010. http://fermat.usach.cl/~msanchez/comprimido/OBJETOS.pdf.

ANEXOS

Anexo 1. Entrevista.

- ✓ A su consideración cuáles son los motivos por los cuales es necesario implementar un componente para la protección de software.
- ✓ ¿Cuál es la situación actual con los productos del departamento una vez que son desplegados?
- ✓ ¿Cuáles son las agravantes que se podrían generar una vez que el producto haya sido desplegado?
- ✓ ¿Cuáles son los aspectos que usted considera debería tener un sistema para la protección se software?
- ✓ ¿En qué aspecto debería enfocarse el sistema para la protección de software?

La entrevista realizada arrojó los siguientes resultados:

El personal entrevistado dejó claro que una vez que los productos son liberados pueden existir inconvenientes, ya que personas deshonestas dentro de la misma institución podrían copiar el producto y utilizarlos en ambientes no autorizados, que también era muy importante contar con una solución propia del departamento que asegurara las aplicaciones de escritorio para evitar este tipo de delitos, y que una solución como esta brindaría una protección lo bastante segura como para disminuir el delito que representa la copia ilegal del software.

Otros aspectos en los que hicieron énfasis fue en la creación de una llave de licencia única tomando características únicas del hardware de la PC donde se quiere instalar el producto, para así evitar la copia ilegal, además enfatizaron que era muy importante el chequeo del periodo activo de la licencia, para evitar utilizar la aplicación fuera del mismo. Otro aspecto destacado por los entrevistados fue que el sistema debería ser capaz de avisar con anterioridad el vencimiento de la licencia, para que el usuario tenga tiempo de poder gestionar otra llave de licencia.

A grandes rasgos esta fueron las consideraciones más importantes a tomar en cuenta para llevar a cabo el desarrollo de esta solución, la entrevista realizada tributó a la redacción de la situación problemática del presente documento, así como a la obtención de los requisitos funcionales.

GLOSARIO DE TÉRMINOS

Términos	Definiciones
CASE	Ingeniería de Software Asistida por Ordenador (Computer Aided Software Engineering), Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.
IDE	Entorno Integrado de Desarrollo (Integrated Develop Environment), es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica.
RUP	Proceso Unificado Racional (Rational Unified Process), proceso de desarrollo de software, es la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.
UML	Lenguaje Unificado de Modelado (Unified Modeling Language), Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.