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

Título: Herramienta para la gestión de repositorios de software para los servidores del departamento Nova.

Autor:

Tomás Peña Cabrales.

Tutores:

Ing. Miguel Albalat Águila.

Ing. Jorge Luis Machin Castillo.

Ciudad de La Habana, Cuba 2012

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Tomás Peña Cabrales

_____

Firma del autor

Ing. Miguel Albalat Águila

_____

Firma del tutor

Ing. Jorge Luis Machin Castillo

_____

Firma del tutor

Agradecimientos

A mi familia por ser lo mejor que he descubierto y siempre apoyarme en cuanto sueño he realizado, espero algún día ser merecedor de todo lo que me han dado.

A mi amigo “el curro López”, el hermano que no tuve y mi maestro. ¿Por qué no?

A Sandry mi novia de la universidad, la que nunca tuve.

A mis hermanos de cinco años de estudio y madrugadas los quiero, sin excepción de alguno.

A mis profesores que han sido educadores y excelentes amigos. Migue, Firvida, Machin, Dariem, Miranda, Daniel, “el puro” y Román, espero haber estado a la altura de su compañía y enseñanza.

A mi país y mi revolución que no dejó de darme lecciones y consuelo cuando estuve lejos.

A todos los que de una forma u otra me han hecho alguien querido.

Muchas gracias...

Dedicatoria

A quienes Bertolt Brecht llamara imprescindibles...

Resumen

En el departamento Nova existe la necesidad de una herramienta que gestione los repositorios y a su vez permita realizar con estos tareas que hasta el momento resulta difícil completarlas. Para darle solución a esta necesidad existente se realizó un estudio a nivel nacional e internacional de la documentación referente al tema, en busca de herramientas disponibles para ser usadas. Se analizó y diseñó una herramienta de gestión para repositorios de software que permitiera su posterior implementación. Al finalizar el proceso de desarrollo de la herramienta se realizaron pruebas unitarias y de aceptación para validar la solución obtenida. Se utilizó la metodología ágil de desarrollo conocida como OpenUp, Eclipse como entorno de desarrollo para el lenguaje de programación Python. Una vez implementada la solución propuesta, se obtuvo una herramienta de gestión de repositorio en los servidores del departamento Nova, la cual es capaz de construir y poner a disposición de sus clientes los repositorios de manera inmediata, también cumple con los requisitos propuestos. La misma es un aporte al proceso de desarrollo del departamento Nova así como al proceso de migración a Software Libre y aplicaciones de código abierto que se realiza actualmente en el país con el objetivo de lograr la independencia tecnológica.

PALABRAS CLAVE

Paquete, paquete de *software*, repositorio, repositorio de *software*.

Índice de contenido

Índice de contenido 1
Introducción 1
Capítulo 1: Fundamentación del tema 4
Antecedentes en el departamento Nova 4
Nova Build System 4
1.1 Gestor de repositorios de software 5
1.1.1 Createrepo 5
1.1.2 Debmirror 6
1.1.3 Reprepro 7
1.2 Autenticación y firmas de paquetes 8
1.2.1 PGP 8
1.2.2 GnuPG 9
1.3 Lenguaje de Programación 9
1.3.1 C++ 10
1.3.2 Python 10
1.3.3 Perl 11
1.4 Plataforma de desarrollo 12
1.4.1 Geany 12
1.4.2 NetBeans 13
1.4.3 Eclipse 13
PyDev 13
PyUML 13
Epic 13
1.5 Base de Datos 14
1.5.1 MySQL 14
1.5.2 PosgreSQL 15
1.5.3 SQLite 15
1.6 Metodología de desarrollo 16

1.6.1 OpenUP 17
1.7 Propuesta 18
Capítulo 2: Diseño del sistema 21
Introducción 21
2.1 Actores del sistema 21
2.2 Requisitos Funcionales 21
2.3 Requisitos no funcionales 25
2.3.1. Apariencia o interfaz externa 25
2.3.2 Software 25
2.4 Diagrama de casos de uso del sistema 25
2.5 Descripción de los casos de usos del sistema 26
2.7 Diagrama de vista lógica 28
2.8 Diagrama de Interacción 29
2.8.1 Diagrama de colaboración 29
2.9 Patrones de diseño 29
2.9.1 Patrones GRASP 30
2.9.1.1 Experto en información 30
2.9.1.2 Creador 30
2.9.1.3 Alta cohesión 31
2.9.1.4 Bajo acoplamiento 31
2.10 Conclusiones del capítulo 31
Capítulo 3: IMPLEMENTACIÓN Y PRUEBA 32
3.1 Diagrama de despliegue 32
3.2 Diagrama de Componentes 33
3.3 Validación funcional 37
3.3.1 Pruebas de software 38
3.3.1.1 Pruebas de caja negra 38
3.3.1.1.1 Pruebas de Validación 38
3.3.1.2 Pruebas de caja Blanca 39
3.3.1.2.1 Pruebas unitarias 39
3.4 Conclusiones del capítulo 39

Conclusiones..... 41
Recomendaciones..... 42
Referencia Bibliográfica..... XLIII
Glosario de términos..... XLVI
Anexos..... XLVIII
Anexo 1 Descripción de casos de uso..... XLVIII
Tabla gestionar repositorio de paquetes de software XLVIII
Tabla gestionar llave de mantenedor..... L
Tabla gestionar mantenedor de paquetes..... LI
Tabla Chequear estado de repositorio..... LII
Tabla firmar repositorio de paquetes de software..... LIV
Tabla mezclar repositorio de software..... LV
Anexo 2: Diagrama de casos de uso..... LVIII
Anexo 3: Diagrama de clases del sistema..... LIX
Anexo 4: Diagrama de vistas lógicas..... LX
Anexo 5: Diagrama de componentes..... LXI
Anexo 6: Descripción de variables..... LXII
Anexo 7: Diseños de casos de prueba..... LXIV
7.1 Crear repositorio..... LXIV
7.2 Mezclar repositorio..... LXV
7.3 Eliminar repositorio..... LXVI
7.4 Actualizar repositorio..... LXVI
7.5 Crear mantenedor..... LXVII
7.6 Eliminar mantenedor..... LXVII
7.7 Chequear integridad de repositorios..... LXVIII
Anexo 8: Diagramas de colaboración..... LXIX
8.1 Mezclar repositorio..... LXIX
8.2 Crear repositorio..... LXIX
8.3 Eliminar mantenedor..... LXX
8.3 Crear mantenedor..... LXX

Introducción

Un rasgo distintivo de los sistemas operativos es lo intuitivo de sus herramientas de trabajo. Otro punto importante es que los usuarios de dichos sistemas necesitan disponer de nuevas actualizaciones y *software*, sintiéndose parte de la continua innovación tecnológica.

Un paquete de *software* es un fragmento de código o funcionalidad, el cual se puede encontrar compilado en forma de binario o texto legible como código fuente. En los Sistemas Operativos GNU\Linux existe una estructura llamada repositorio de *software*, la cual es piedra angular de las continuas mejoras en las que trabaja su comunidad de desarrollo, ya que son los encargados de brindar paquetes de software o aplicaciones certificadas que son probadas y compatibles con estas distribuciones. Un mal funcionamiento de los repositorios traería consigo un gran descontento a los usuarios del sistema al cual dicho repositorio de *software* brinda paquetes, ya que no dispondrían de actualizaciones tanto de seguridad como de nuevas soluciones a los problemas detectados por la comunidad. Una gestión eficaz de los repositorios para un proyecto GNU\Linux se traduce en clientes más afianzados a sus productos así como credibilidad ante la comunidad.

En nuestro país se hacen grandes esfuerzos para migrar a plataformas libres y aplicaciones de código abierto la infraestructura informática en aras de alcanzar la independencia tecnológica. Esta idea dio a luz el departamento Nova, ubicado en la Universidad de las Ciencias Informáticas (UCI). Este proyecto surge con la intención de brindar un apoyo a la progresiva migración de Cuba al *software* libre desarrollándose allí un sistema operativo (SO) GNU/Linux bajo el mismo nombre. Nova debido a un cambio estratégico realizado en el a principios del año 2011 utiliza como base la versión de Ubuntu 10.04 LTS, la cual contó con un amplio soporte brindado por la empresa Canonical[1] y su comunidad de desarrollo.

1 **Canonical Ltd.** es una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth, para la creación de proyectos relacionados con *software* libre y soporte. Canonical tiene su sede principal en Londres, Reino Unido, pero sus empleados se encuentran en varias partes del mundo.

En el departamento Nova el manejo de los repositorios de *software* se hace a través de varias herramientas que realizan parcialmente este trabajo de forma manual y engorrosa, existiendo cierta desorganización y desconocimiento en lo que se debe hacer para completar la correcta gestión de dichos repositorios. Procesos comunes, como el de mezclar repositorios, carecen de herramientas que lo faciliten y lo automaticen. Además se evidencia la falta de una interfaz gráfica o algún mecanismo que facilite su elaboración. Se necesita también una interfaz de comunicación para su integración con otras herramientas que se desarrollen en el proyecto.
Una vez planteado lo anterior se define el siguiente **problema científico**: ¿Cómo facilitar a los desarrolladores el proceso gestión en los repositorios de paquetes de *software* en los servidores del departamento Nova?

En este trabajo se considera como **objeto de estudio** la gestión de repositorios, con un **campo de acción** enfocado a las herramientas de gestión de repositorio en el departamento Nova.

Se trazó como **objetivo principal** desarrollar una aplicación para la gestión de repositorios de *software* en los servidores del departamento Nova, orientado a los desarrolladores del departamento Nova.

Partiendo del objetivo principal, se plantean los siguientes **objetivos específicos**:

1. Consultar documentación especializada sobre las herramientas para la gestión de repositorios de *software*.
2. Analizar y diseñar una herramienta para la gestión de repositorios de *software* .
3. Implementar una herramienta para la gestión de repositorios de *software.*
4. Realización de pruebas unitarias y de aceptación a la herramienta desarrollada para la gestión de repositorios de *software*.

Esta investigación tiene como **idea a defender** que una aplicación de gestión de repositorios de *software* para los servidores del departamento Nova traerá consigo una posible aceleración en el proceso de desarrollo.

El proceso estará dirigido por las siguientes **tareas de investigación**:

1. Análisis del funcionamiento de los sistemas de gestión de repositorio de *software*.
2. Obtención de una propuesta de solución.
3. Construcción de una herramienta de gestión integral de repositorio de *software*.
4. Pruebas a la herramienta desarrollada.

Los **métodos** usados fueron:

Analítico-sintético, permitió la realización de análisis de distintas herramientas de gestión de repositorios de *software*, para una mejor comprensión de sus características.

Capítulo 1: Fundamentación del tema

Los sistemas de gestión de paquetes son utilizados por los sistemas GNU/Linux para manejar el proceso de instalación y/o desinstalación de aplicaciones y bibliotecas. Estos sistemas de gestión obtienen los paquetes desde alguna ubicación específica que puede encontrarse disponible a través de la red o en dispositivos físicos como discos duros, compactos o cualquier dispositivo de almacenamiento extraíble. A estas ubicaciones se les conoce como repositorios de paquetes de *software*. Existen distintas configuraciones estructurales para estos repositorios, en dependencia de qué sistema de gestión se use, pero todas coinciden en que pueden encontrarse los paquetes de *software* disponibles para instalar en la distribución que los mantenga. Otro detalle a tener en cuenta es que los paquetes incluidos en los repositorios pueden ser de código fuente o precompilados. [5.]

La gestión de un repositorio de *software* consiste en un proceso de creación, configuración, manejo de paquetes y mantenimiento de los paquetes de *software* e índices de dicho repositorio de *software*. Un índice no es más que un fichero de texto donde se encuentran reflejados datos detallados de cada paquete contenido en el repositorio potenciando así la velocidad de búsqueda de paquetes para su posterior descarga e instalación.

En el presente trabajo se manejará el término “mantenedor”, que será la persona responsable del mantenimiento de paquetes de *software* que le están asignados.

Antecedentes en el departamento Nova.

En el departamento Nova existen herramientas elaboradas por parte de los especialistas las cuales resuelven las necesidades más básicas de la producción. A continuación hacemos referencia a ellas.

Nova Build System

Este consiste en un conjunto de *script*[2] en 309,4 Kb de ficheros ,escritos en lenguaje Bash, usados para realizar la gestión de repositorios en el departamento Nova.

Este programa se ejecuta en una consola de línea de comandos, donde un especialista realiza la gestión de repositorio, se entiende por gestión de repositorio la creación, manejo de paquetes y mantenimiento. Utiliza la herramienta Reprepro como dependencia para el manejo de los repositorios bajo su gestión. Para el uso de esta herramienta se requiere un conocimiento avanzado de la tecnología empleada por parte del administrador de los repositorios.

1.1 GESTOR DE REPOSITORIOS DE *SOFTWARE*

Existen herramientas especializadas en el manejo de repositorios y descarga de paquetes los cuales se pudieran usar para realizar el manejo de los repositorios facilitando así la gestión. Teniendo en cuenta una que la herramienta buscada debe cumplir varios requisitos como el permitir el firmado de repositorios usando firma digital se realiza el siguiente estudio.

1.1.1 CREATEREPO

Herramienta de gestión de repositorios de paquetes de *software* bastante conocida. Posee como rasgo distintivo que su gestión es exclusiva de repositorios de paquetes .rpm[3] para las distribuciones *Red Hat*[4] o que tengan a esta como origen, entre las cuales es utilizado. Es capaz de crear repositorios de software basándose en la información obtenida a partir del XML de los metadatos rpm.

2 **Script:** archivo de órdenes o archivo de procesamiento por lotes, vulgarmente referidos como "script", es un programa usualmente simple, que por lo regular se almacena en un archivo de texto plano.

3 **RPM:** Originalmente llamado ***Red Hat Package Manager***, pero se convirtió en acrónimo recursivo. Es una herramienta de administración de paquetes pensada básicamente para GNU/Linux. Es capaz de instalar, actualizar, desinstalar, verificar y solicitar programas. RPM es el formato de paquete de partida del Linux *Standard Base*.[16.]

4 **Red Hat Linux**. Una de las primeras distribuciones comerciales de GNU/Linux muy conocida y se puede considerar como la "canónica" de entre las distribuciones comerciales. El trabajo de los distribuidores está básicamente relacionado con tareas de integración y no tanto con el desarrollo de *software*.[17.]

Existen versiones para los sistemas operativos Debian, Ubuntu, *Red Hat* y otros. Presenta yum-utils y Python 2.4 o superior como principales dependencia para su funcionamiento. Además, se encuentra en su versión estable 0.4.11 y una versión 0.9.7 en desarrollo según su sitio oficial. Emplea una base de datos SQLite en el manejo de datos obtenidos a partir de los paquetes de software manejados.

1.1.2 DEBMIRROR

Herramienta utilizada mediante líneas de comandos que permite la creación parcial o total de un repositorio de *software* para distribuciones de Debian o Ubuntu. El mismo, descarga un "*mirror*" o espejo local de todos los paquetes del servidor principal admitiendo cualquier combinación de arquitecturas, distribuciones y secciones. Escrita en lenguaje Perl. Los archivos son transferidos mediante ftp[5], http[6], hftp[7] o rsync[8] y es totalmente compatible con todos los repositorios de *software* oficiales. Cuenta con una interfaz de Python que utiliza un archivo de configuración XML para generar un "mirror" de Ubuntu, este módulo es llamado Mirrorkit. También es capaz de manejar cifrados en md5[9], sch1 y sch256 y los más conocidos formatos de compresión de archivo.

Posee integración con otros *software* necesarios para la gestión de repositorios de *software* como es el sistema de código abierto para autenticación y gestión de llaves de GnuPG. Además usa el módulo de algoritmo rsync para copias remotas. [2]

5 **FTP** (siglas en inglés de ***F**ile **T**ransfer **P**rotocol*, 'Protocolo de Transferencia de Archivos') en informática, es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (*Transmission Control Protocol*), basado en la arquitectura cliente-servidor.[18.]

6 ***Hypertext Transfer Protocol*** o **HTTP** (en español *protocolo de transferencia de hipertexto*) es el protocolo usado en cada transacción de la World Wide Web.

7 **HFTP:** Es un protocolo el cual facilita el acceso a un recurso FTP a través de un proxy HTTP.

8 **Rsync:** Es una aplicación libre para sistemas de tipo Unix y Microsoft Windows que ofrece transmisión eficiente de datos incrementales, que opera también con datos comprimidos y cifrados.

9 **MD5** (abreviatura de *Message-Digest Algorithm 5,* Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits ampliamente usado.

Este *software* solo realiza las labores de descarga y actualización de repositorio, se encuentra muy limitado para realizar tareas con los paquetes y los repositorios una vez descargados.

1.1.3 REPREPRO

Anteriormente llamado *Mirrorer,* este *software* de línea de comandos, especializado en el manejo de repositorios de paquetes de *software* para dpkg[10] es decir, usa fundamentalmente paquetes precompilados .deb[11] , udeb o fuentes .dsc[12]. Está escrito en lenguaje C y se encuentra disponible en su versión 4.8.1.

Presenta la capacidad de almacenar ficheros ya sea que se incluyan manualmente o descargados de algún repositorio de *software*. Los almacena en un directorio ./pool ordenado alfabéticamente en un árbol concebido para este fin. Gestiona los ficheros de configuración y los paquetes de *software* a través de una base de datos libdb4.3 o libdb4.4, según se compile Reprepro por lo que no se necesita un gestor de base de datos convencional. [1]

Es compatible con varios algoritmos para la encriptación de datos como sch1, sch256 y md5. Además, se integra a otros *software* necesarios para la gestión de repositorio como es sistema libre para autenticación GnuPG y los algoritmos usados en la compresión de datos. Emplea llaves generadas por GnuPG para hacer posible la autenticación de los repositorios y los paquetes. Es compatible con los protocolos de trasferencia de datos http, ftp y hftp. También se puede configurar para que funcione mediante un proxy.

Realiza la gestión de repositorio no limitándose solo a la descarga de los paquetes y creación de la estructura del repositorio, sino que es capaz de realizar la administración y manejo de los índices de los repositorios ubicados de forma local en el disco duro. Una vez creado un repositorio hace posible su gestión de forma variada y dinámica.

10 **dpkg:** Es el encargado de desempaquetar e instalar físicamente los archivos de paquetes Debian(archivos con la extensión .deb).

11 **.deb:** es la extensión del formato de paquetes de *software* de Debian y derivadas (ej.Ubuntu), y el nombre más usado para dichos paquetes.

12 **.dsc:** Fichero de control de código fuente. Es generado por dpkg al generar el archivo origen.

Reprepro cuenta con un número de ficheros que hacen posible su fácil configuración tornándose en una versátil y potente herramienta. Permite la existencia de varias versiones del mismo paquete de software en la misma base de datos, esto sólo si se desea. Una característica que puede ser explotada es el contar con comandos cortos si se compara con otras herramientas homólogas, facilitando así la implementación de una API o manejador que interactué con este a modo de interfaz, simplificando la progresiva implementación de la solución al abstraer al desarrollador de las especificidades de la herramienta.

Como dificultad con que cuenta el *software* en cuestión, se tiene que la base de datos con que este realiza sus operaciones es propensa a corromperse cuando el manejo de los repositorios no es el correcto. De emplearse este manipulador de repositorios en nuestra solución se deberá que tener en cuenta dicha característica.

1.2 Autenticación y firmas de paquetes.

Se realiza un estudio en busca de una herramienta capaz de generar firmas digitales necesarias para la gestión de repositorio. Dicho estudio queda reflejado a continuación.

1.2.1 PGP[13]

Combina algunas de las mejores características de la criptografía simétrica y la criptografía asimétrica para clasificarse como un criptosistema híbrido.

Cuando un usuario emplea PGP para cifrar un texto plano, dicho texto es comprimido. La compresión de los datos ahorra espacio en disco, tiempos de transmisión y, más importante aún, fortalece la seguridad criptográfica. La mayoría de las técnicas de criptoanálisis explotan patrones presentes en el texto plano para craquear el cifrado. La compresión reduce esos patrones en el texto plano, aumentando enormemente la resistencia al criptoanálisis.

13 **PGP:** Pretty Good Privacy, es el sistema de codificación por excelencia desde su creación. Ampliamente usado en el mundo sobre todo en el campo de la seguridad en las nuevas tecnologías de la información

De sus algoritmos más usados está Triple-DES, que consiste en un proceso triple de encriptación con una misma llave privada del algoritmo DES[14].

1.2.2 GnuPG

GPG es una herramienta de cifrado y firmas digitales, que viene a ser un remplazo del PGP *(Pretty Good Privacy)* pero con la principal diferencia que es *software* libre licenciado bajo la GPL.[3]

GnuPG o *GNU Privacy Guard* por sus siglas en inglés, es una herramienta de GNU para asegurar comunicaciones y almacenamiento de datos. Se puede utilizar para cifrar los datos y crear firmas digitales. Incluye un sistema avanzado de gestión de claves y cumple con el estándar OpenPGP descrito en el estándar RFC-4880. Como tal, está destinado a ser compatible con PGP del PGP Corp. y otras herramientas de OpenPGP. [6.]

El *software* es capaz de codificar y decodificar los algoritmos de encriptación md5, sch1 y sch256, muy usados para la validación de sumas de chequeo y firma de paquetes. Se encuentra integrado con las herramientas de manipulación de repositorios y al sistema operativo en general, característica la cual podría ser aprovechada cuando se proponga una solución.

1.3 Lenguaje de Programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático.[24.] Permite al programador especificar de manera precisa sobre qué datos debe operar una computadora, cómo estos datos deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. [25.]

Los lenguajes de programación y los entornos de desarrollo integrados tienen suma importancia para cualquier proyecto de software pues con estos se han creado

14 **DES:** Algoritmo de cifrado muy usado en la actualidad. Se considera uno de los algoritmos más difícil de forzar de los existentes. En la actualidad solo se ha logrado romper su seguridad en contadas ocasiones.

innumerables programas y herramientas que han ayudado al hombre a controlar de una manera más sencilla a los ordenadores, así como realizar múltiples tareas y actividades. En la actualidad existe un numeroso conjunto de lenguajes que se utilizan en el desarrollo de aplicaciones, algunos de estos son mencionados a continuación.

1.3.1 C++

Se considera un lenguaje robusto que abarca varios paradigmas incluyendo la programación orientada a objetos. A pesar de brindar la posibilidad de trabajar tanto a bajo nivel como a alto nivel, es uno de los que menos automatismo trae, lo que dificulta su aprendizaje. Es un lenguaje portable que no se limita a una arquitectura de hardware específica, además de ser multiplataforma, también brinda la posibilidad de crear aplicaciones modulares.

En este lenguaje se encuentran implementadas la mayoría de las herramientas para el trabajo de repositorios. En caso de que se utilizara en la implementación de una solución se obtendría un código de muy rápida ejecución pero de un lento proceso de aprendizaje. Además, se dificultaría su posterior mantenimiento.

1.3.2 Python

Python es un lenguaje interpretado de programación, multiplataforma, flexible, su implementación está bajo la licencia de código abierto Python Software Foundation License.

Python, al ser un lenguaje interpretado, implica ahorro de tiempo durante el desarrollo de un programa ya que no necesita de compilación. El intérprete puede usarse de modo interactivo, el cual hace fácil el experimentar con las características del lenguaje para probar funciones durante la etapa inicial de desarrollo. [23.]

Permite escribir programas muy compactos y legibles, permitiéndole ser más pequeños que sus contrapartes en C u otros lenguajes por las siguientes razones:

1. **Lenguaje Interpretado o de Script:** es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje de máquina.

2. **Funciones y bibliotecas:** dispone de un gran cúmulo de funciones incorporadas en el propio lenguaje, permitiendo el tratamiento de cadenas de caracteres, números, archivos, etc. Además, existen librerías que son importadas con facilidad en los programas para tratar temas específicos.
3. **Detección dinámica del tipo de variable en tiempo de ejecución:** se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo de valor al que se asigne, y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo.
4. **Sintaxis clara:** posee una sintaxis visual gracias a la notación identada (con márgenes) de obligado cumplimiento. Además, para distanciar las porciones de código se aconseja tabular, colocando un margen al código que iría dentro de una función o un bucle. Esto beneficia a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.
5. **Multiplataforma:** el intérprete de Python está disponible en multitud de plataformas (Solaris, GNU/Linux, DOS, Window, OS/2, Mac OS) por lo que si no se utilizan bibliotecas específicas de cada plataforma el programa podrá correr en todos estos sistemas sin grandes cambios.
6. **Orientado a Objeto:** la orientación a objetos es un paradigma de programación en el que los conceptos relevantes del mundo real para los problemas se trasladan a clases y objetos del programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

1.3.3 PERL

Es un lenguaje de propósito general originalmente desarrollado para la manipulación de texto y que actualmente es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red, desarrollo de interfaz gráfica de usuario, entre otras. Entre sus principales características se tiene que es muy fácil de usar, soporta

tanto la programación estructurada como la programación orientada a objeto y la funcional, tiene incorporado un poderoso sistema de procesamiento de texto y una enorme colección de módulos disponibles. Perl es software libre y está licenciado bajo la Licencia Artística y la GNU(*General Public License)*. Además es soportado actualmente por la mayoría de las plataformas existentes [4].

Perl está instalado por defecto en las distribuciones más populares de GNU/Linux incluyendo Gentoo, Slackware, Mandriva, Debian, RedHat y SUSE.

1.4 Plataforma de desarrollo.

La implementación es una etapa clave en el proceso de desarrollo y será necesaria toda la ayuda posible en el uso de los estándares de código y corrección de todo tipo de errores como también la posibilidad poder interpretar el código implementado sin necesidad de compilarlo. Es en estos aspectos donde cobra importancia el uso de los entornos integrados de desarrollo (IDE por sus siglas en inglés).

1.4.1 Geany

Editor de texto construido con GTK2[15], tiene características básicas de un entorno integrado de desarrollo. Es muy ligero y carece de un número excesivo de dependencias. Es compatible con gran cantidad de formatos. Algunas de sus bondades son el uso de sintaxis inteligente, identación de código, autocompletado de símbolos, inclusión de nuevos *plug-in*, cierre automático de etiquetas XML y HTML, soporte a los lenguajes C, Java, PHP, HTML, Python, Perl, Pascal, etc... [7.]

Soportado por la mayoría de los sistemas operativos incluyendo Microsoft Windows y se encuentra licenciado bajo GPL.

15 **GTK: The GIMP Toolkit** es un conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario (GUI), principalmente para los entornos gráficos GNOME, XFCE y ROX aunque también se puede usar en el escritorio de Windows, MacOS y otros.

1.4.2 NetBeans

Herramienta libre de código abierto, el entorno de desarrollo NetBeans es conocido por sus bondades entre sus homólogos de GNU/Linux. Brinda reconocimiento de la gramática en el código, así como análisis léxico y semántico en tiempo real. Utiliza un desarrollo modular por lo que se le agregan nuevos soportes a lenguajes y herramientas de programación fácilmente. Se integra con el sistema de control de versiones SVN.

1.4.3 Eclipse

Mediante el uso de *plug-ins*, la plataforma Eclipse incluye una gran variedad de funcionalidades utilizadas a lo largo de todo el ciclo de desarrollo de cualquier solución. A continuación se describen algunos *plug-ins* muy utilizados:

PyDev

Posibilita el desarrollo con el lenguaje Python, permitiendo utilizar todas sus bibliotecas. Incluye reconocimiento de la gramática en el código, reconocimiento de sintaxis, análisis del código, refactorización17[16], debugger18[17], consola interactiva, etc.

PyUML

Permite construir diagramas UML o generarlos a partir de código Python. También es posible generar código Python a partir de diagramas UML.

Epic

Permite el desarrollo con el lenguaje Perl y todas las bibliotecas de este. Cuenta con reconocimiento y de la gramática en el código, reconocimiento de sintaxis, etc. El plug-in también añade puntos de vista adicionales gracias a la información relacionada que permite ver la documentación también permite ejecutar sus aplicaciones Perl y depurar el código.

16 La **refactorización** (del inglés *Refactoring*) es una técnica de la ingeniería de *software* para reestructurar un código fuente, alterando su estructura interna sin cambiar su comportamiento externo.

17 Un **depurador** (en inglés, **debugger**), es un programa usado para probar y depurar (eliminar los errores) de otros programas (el programa "objetivo").

1.5 Base de Datos.

Para lograr consistencia y persistencia en los datos es necesario el empleo de bases de datos que respondan las necesidades de disponibilidad, estabilidad y velocidad de acceso.

La gestión de repositorios de paquetes de *software* requiere características como gestionar bases de datos de forma local en forma de ficheros, capaz de atender gran cantidad de consultas en un corto periodo de tiempo y de forma concurrente aprovechando lo mejor posible los recursos del servidor. También debe caracterizarse por su estabilidad y fiabilidad en el manejo de datos. El gestor empleado con este fin debe cumplir con estas características si se desea obtener cualidades deseables y en el software.

En función de las apreciaciones anteriormente expuestas se realiza el siguiente estudio de varios gestores de base de datos.

1.5.1 MySQL

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Escrito en C y en C++. Probado con un amplio rango de compiladores diferentes. Funciona en diferentes plataformas. Usa GNU Automake, Autoconf, y Libtool para portabilidad. API's[18] disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, y Tcl. Uso completo de multi-threaded mediante threads del kernel. Pueden usarse fácilmente múltiples CPUs (Unidad Central de Procesamiento por sus traducción al español) si están disponibles. Proporciona sistemas de almacenamientos transaccionales y no transaccionales. Usa tablas en disco B-tree (`MyISAM`) muy rápidas con compresión de índice. Relativamente sencillo de añadir otro sistema de almacenamiento. Esto es útil si desea añadir una interfaz SQL para una base de datos propia. Un sistema de reserva de memoria muy

18 **Interfaz de programación de aplicaciones** o API (del inglés *Application Programming Interface*) 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.

rápido basado en threads. Joins muy rápidos usando un multi-join de un paso optimizado. Tablas hash en memoria, que son usadas como tablas temporales. Las funciones SQL están implementadas usando una biblioteca altamente optimizada y deben ser tan rápidas como sea posible. Normalmente no hay reserva de memoria tras toda la inicialización para consultas. El código MySQL se prueba con Purify (un detector comercial de memoria perdida) así como con Valgrind, una herramienta GPL. El servidor está disponible como un programa separado para usar en un entorno de red cliente/servidor. También está disponible como biblioteca y puede ser incrustado (linkado) en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible. [9.]

1.5.2 POSGRESQL

Es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD. Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG (*PostgreSQL Global Development Group*).

Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo "*commit*". Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos. PostgreSQL provee nativamente soporte para números de precisión arbitraria, cadenas de texto con un largo ilimitado, figuras geométricas (con una variedad de funciones asociadas), direcciones IP (IPv4 e Ipv6), bloques de direcciones estilo CIDR, direcciones MAC, "*arrays*". [10.]

Este gestor cuenta con API's para variedad de lenguajes entre los que se encuentran Java, Python, C++,C#, etc...

1.5.3 SQLite

SQLite es un sistema completo de bases de datos que soporta múltiples tablas, índices, "*triggers*" y vistas. No necesita un proceso separado funcionando como servidor ya que lee y escribe directamente sobre archivos que se encuentran en el disco duro. El formato de la base de datos es multiplataforma e indistintamente se puede utilizar el mismo archivo en sistemas de 32 y 64 bits. La base de datos se almacena en un único fichero a diferencia de otros DBMS que hacen uso de varios archivos. SQLite emplea registros de tamaño variable de forma tal que se utiliza el espacio en disco que es realmente necesario en cada momento. El código fuente está pensado para que sea entendido y accesible por programadores promedio. Todas las funciones y estructuras están bien documentadas.

También cuenta con las siguientes características:

- Restricciones *"FOREIGN KEY"*.
- Soporte completo para *triggers* (disparadores).
- Soporte completo para *"ALTER TABLE"*.
- Solamente implementa las instrucciones *"RENAME TABLE"* y *"ADD COLUMN"*.
- Sólo está implementada la instrucción *"LEFT OUTER JOIN".*
- Se posibilita escribir en *"VIEWs"*, ya que las vistas en SQLite son de sólo lectura
- Desaparece "*GRANT*" y "*REVOKE*", pues no tienen sentido en un sistema de bases de datos embebido [11.].

Este gestor cuenta con API's para variedad de lenguajes entre los que se encuentran Java, Python, C++,C#, entre otros. Es muy usado en la actualidad gracias a sus ventajas sobre otros gestores.

1.6 Metodología de desarrollo.

Es un conjunto de pasos y procedimientos que deben seguirse para desarrollar *software*. Utilizada para dividir un proyecto en etapas, desarrollando las tareas que se llevan a cabo en cada etapa y las restricciones que deben aplicarse en cada una de ellas. En estas metodologías se especifican las técnicas y herramientas que se deben emplear para controlar y gestionar un proyecto. [12.]

Actualmente, existen diferentes metodologías y técnicas para el desarrollo de productos las que fueron analizadas y definidas en la arquitectura del proyecto, quedando como propuesta final la metodología OpenUP. [13.]

1.6.1 OpenUP

OpenUP/Basic es un *Framework* de procesos de desarrollo de programas de código abierto, que con el tiempo espera cubrir un amplio conjunto de necesidades en el campo del desarrollo de programas. Permite un abordaje ágil al programa en análisis con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas. Es un proceso interactivo de desarrollo de *software* simplificado, completo y extensible; para pequeños equipos de desarrollo, que valoran los beneficios de la colaboración y de los involucrados, con el resultado del proyecto, por encima de formalidades innecesarias. [14.]

Esta metodología es la utilizada en el departamento Nova ya que permite detectar errores tempranos a través de un ciclo iterativo, y también por ser una metodología ágil, que tiene un enfoque centrado en el cliente.

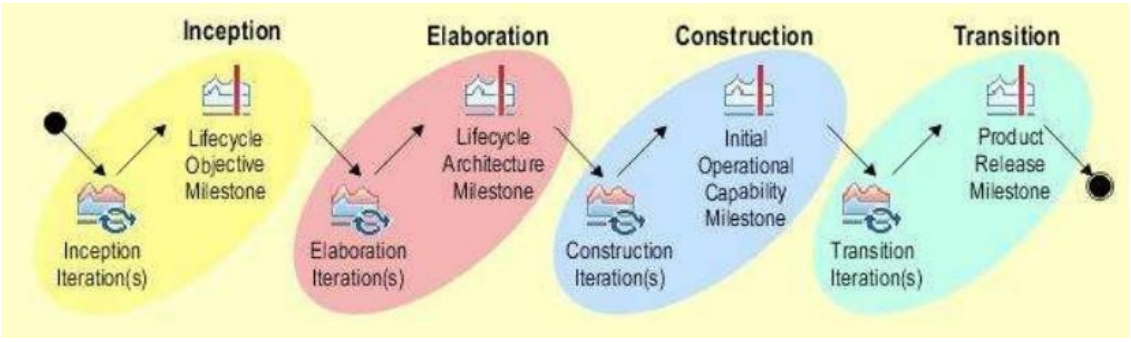

Figura 1: Ciclo de vida OpenUP

En la figura 1 se pueden observar las cuatro fases de desarrollo que propone la metodología: la primera es la fase de Concepción, en donde los objetivos principales son:

- Realizar el planteamiento del problema.
- Elaborar la justificación.
- Definir objetivo general y objetivos específicos.
- Obtener toda la información relacionada con el proyecto.
- Capturar las necesidades de los clientes en los objetivos del ciclo de vida para el proyecto.

La segunda fase es la de Elaboración. Esta fase tiene como objetivo:

- Definir los riesgos significativos para la arquitectura.
- Establecer la base para la elaboración de la arquitectura del sistema.

La tercera fase es Construcción. Esta fase tiene como objetivo:

- Diseñar, implementar y realizar las pruebas de las funcionalidades para realizar un sistema completo.
- Completar el desarrollo del sistema basado en la arquitectura definida.

La última fase es la de Transición. Esta fase tiene como objetivo:

- Asegurar que el sistema sea entregado a los usuarios.
- Evaluar la funcionalidad y escena del último entregable de la fase de construcción.

1.7 PROPUESTA

Luego de la investigación precedente, se elabora la siguiente propuesta a desarrollar:

- Como gestor de repositorio se propone Reprepro ya que según la documentación encontrada y conocimiento previo sobre dicha herramienta en el ambiente de producción, posee características que pueden ser usadas en la solución final. Se tomó esta decisión después de valorar las bondades ofrecidas y analizar las

respuestas obtenidas vía correo electrónico a la comunidad de usuarios de esta herramienta mediante una encuesta realizada. Se tubo conocimiento que las bases de datos de esta herramienta no se corrompen con el uso normal para el que fue diseñada, no comportándose de la misma manera cuando el usuario manipula los paquetes arbitrariamente sin que medie la herramienta en cuestión. Este dato fue confirmado por los especialistas del departamento Nova y por pruebas realizadas a la herramienta.

- Como sistema de autenticación se propone usar GnuPG, por su estabilidad e integración a las demás herramientas presentes en el estudio y los sistemas operativos GNU/Linux. Esta herramienta cuenta con abundante documentación y presenta API's para su uso en varios lenguajes de programación.

- Se propone como lenguaje de programación a utilizar Python para el desarrollo de la aplicación dada su característica de ser orientado a objetos, su alto grado de legibilidad y su detección dinámica del tipo de variable en tiempo de ejecución. Posee un desarrollo abierto, por lo que está en un proceso de continuo desarrollo por una gran comunidad de desarrolladores. Aproximadamente cada seis meses se hace pública una nueva versión de Python y posee una variada documentación comenzando por la expuesta en su sitio web, así como sus librerías, todos disponibles en la web de manera gratuita.

- Se propone como plataforma de desarrollo Eclipse, por ser una plataforma modular estable y contar con características deseables para el desarrollo como es el reconocimiento de la lexicografía del código. Posee el módulo como el Pydev que permiten la interpretación del lenguaje Python y es integrable con el sistema de control de versiones SVN.

- Como gestor de base de datos se empleará SQLite aprovechando lo flexible y versátil de su gestión posibilitando la entrada y salida de datos a la base de datos de forma local. El acceso a datos es codificado si así se desea y a una velocidad considerable. Facilitando la portabilidad de su base de datos gestionadas.

- La solución a desarrollar está enmarcada en el departamento Nova y la metodología a utilizar será OpenUP aunque se encuentra en proceso de aprobación por el departamento de calidad de *software*. Se eligió esta metodología entre otros factores debido a que la solución va a ser parte de un sistema más abarcador que usarán esta metodología como es el sistema de compilación distribuido de Nova. Con esta metodología de cubren las necesidades productivas requeridas en la solución.

Capítulo 2: Diseño del sistema

Introducción

En este capítulo se realiza el diseño de la aplicación basada en la metodología OpenUp, la cual establece primeramente la definición de los actores del sistema, posteriormente los requisitos funcionales para la modelación del Diagrama de casos de usos del sistema y, los requisitos no funcionales. Finalmente, propone la modelación del Diagrama de clases del diseño y el Diagrama de interacción por cada caso de uso.

2.1 Actores del sistema

Los actores de un sistema pueden ser a su vez otros *software*, *hardware* externo o personas que interactúan con nuestro producto, ya sea para inicializar una funcionalidad o brindarle información a este. Para el presente trabajo se definió el siguiente actor.

Nombre del Actor	Descripción
Administrador	Administrador o responsable del mantenimiento y administración de los repositorios de paquetes de *software*.

Tabla 1: Actores del Sistema

Observación: En el presente se maneja el término “Mantenedor” que no es más que un actor que no posee una incidencia directa en la herramienta desarrollada pero es necesario realizar su gestión. Este será el responsable de mantener paquetes de software los cuales estarán bajo su supervisión.

2.2 Requisitos Funcionales

Los casos de uso son fragmentos de funcionalidad que brindan utilidad al usuario y junto con los requisitos funcionales definen el producto así como su utilidad.

Los casos de uso modelan el sistema desde el punto de vista del usuario. Son creados una vez definidos los requerimientos funcionales y deben cumplir los siguientes objetivos:

- Definir los requisitos funcionales y operativos del sistema (producto), diseñando un escenario de uso acordado por el usuario final y el equipo de desarrollo.
- Proporcionar una descripción clara y sin ambigüedades de como el usuario final interactúa con el sistema y viceversa.
- Proporcionar una base para la validación de las pruebas.

En el análisis y diseño de la herramienta se detectaron un total de 18 requisitos funcionales del sistema contenidos en 8 casos de uso los cuales se enuncian a continuación.

CU1. Gestionar repositorio de paquetes de *software* fuente.

RF1. Crear repositorio de paquetes de *software* fuente.

RF2. Eliminar repositorio de paquetes de *software* fuente.

RF3. Actualizar repositorio de paquetes de *software* fuente.

CU2. Gestionar repositorio de paquetes de *software* binario

RF4. Crear repositorio de paquetes de *software* binario.

RF5. Eliminar repositorio de paquetes de *software* binario.

RF6. Actualizar repositorio de paquetes de *software* binario.

CU3. Gestionar repositorio de paquetes de *software* Udebs.

RF7. Crear repositorio de paquetes de *software* Udebs.

RF8. Eliminar repositorio de paquetes de *software* Udebs.

RF9. Actualizar repositorio de paquetes de *software* Udebs.

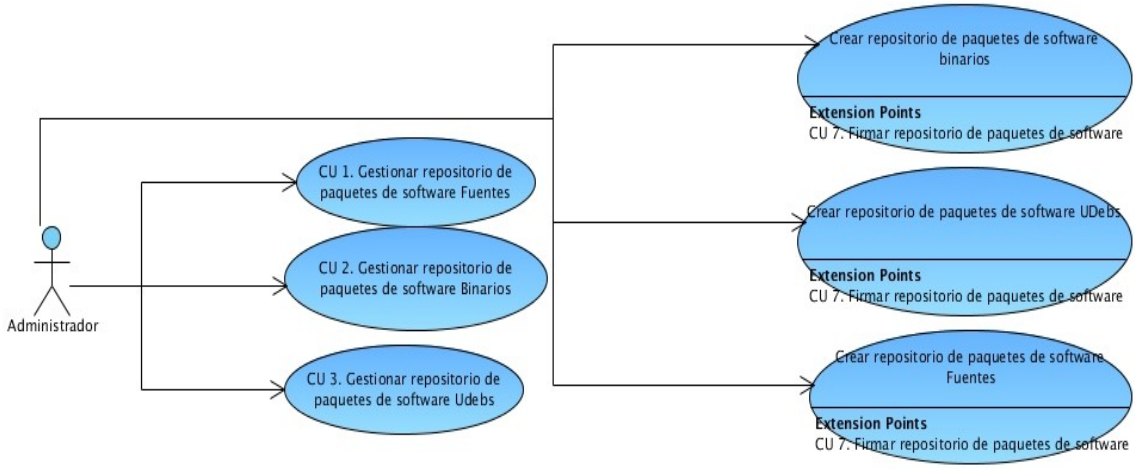

Figura 2: Casos de uso donde se contienen los requisitos del RF1 al RF9.

Los repositorios de cualquiera que fuese su tipo son creados, eliminados y actualizados directamente por el administrador del sistema.

Observación: Los casos de uso CU1, CU2 y CU3 no cuentan con ninguna diferencia en su realización.

CU4. Gestionar mantenedor de paquetes

RF10. Crear mantenedor de paquetes.

RF11. Eliminar mantenedor de paquetes.

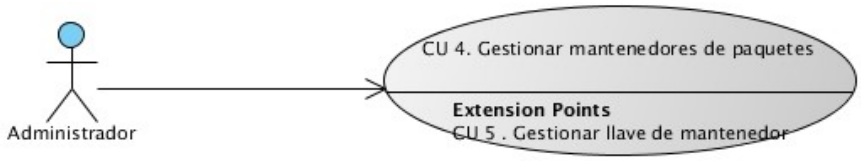

Figura 3: Casos de uso donde se contienen los requisitos RF10 y RF11.

Se gestionan los mantenedores como usuarios que poseerán paquetes de *software* bajo su responsabilidad.

CU5. Gestionar llave de mantenedor.

RF12. Crear llave de mantenedor.

RF13. Eliminar llave de mantenedor de paquetes.

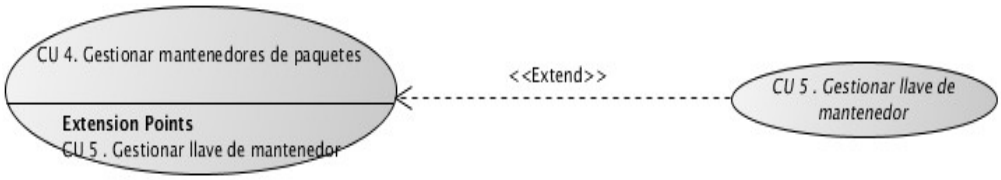

Figura 4: Casos de uso donde se contienen los requisitos RF12 y RF13.

Sincronizado con la creación o eliminación de un mantenedor de paquetes de *software*, está la creación y eliminación de las llaves usadas para firmar paquetes.

CU6. Chequear estado de repositorio.

RF14. Brindar informe de repositorio.

RF15. Alertar mal funcionamiento de repositorio.

RF16. Enviar correo a mantenedor.

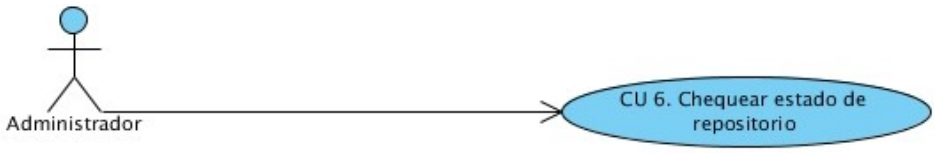

Figura 5: Casos de uso donde se contienen los requisitos RF14, RF15 y RF16.

Los repositorios de *software* por su propia estructura y funcionamiento requieren que se mantenga un correcto manejo de sus índices y se respete su integridad. Para chequear esto se realiza un escaneo del repositorio en busca de errores que son notificados al responsable del repositorio.

CU7. Firmar repositorio de paquetes de *software*.

RF17. Firmar repositorio de paquetes de *software*.

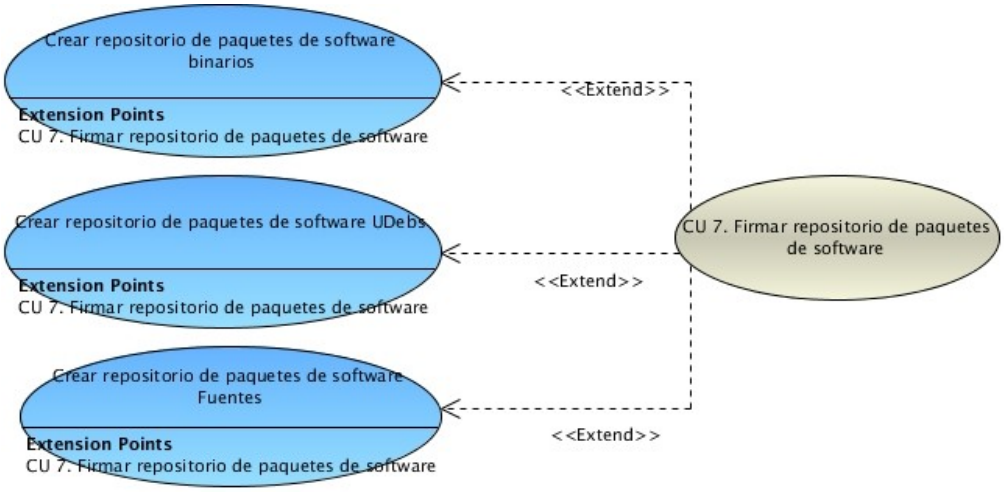

Figura 6: Casos de uso donde se contiene el requisito "firmar repositorio de paquetes de *software*"

Cuando el administrador crea un nuevo repositorio, automáticamente después se le indica a Reprepro la llave perteneciente al mantenedor que inicializa el caso de uso para firmar los índices del repositorio.

CU8. Mezclar repositorios.

RF18. Mezclar repositorios de paquetes de *software.*

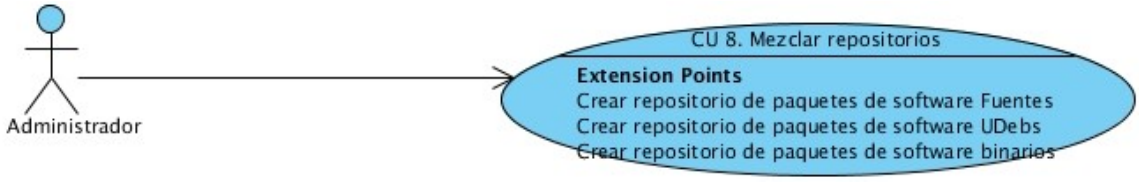

Figura 7: Casos de uso donde se contiene el requisito "Mezclar repositorios de paquetes de *software*"

Este requisito de le permite al administrador del sistema hacer mezclas entre los paquetes de dos repositorios seleccionados y unificarlos en un nuevo repositorio.

2.3 Requisitos no funcionales

Restricciones que afectan a los servicios o funciones del sistema, tales como: restricciones de tiempo, definición de estándares y otras condiciones necesarias para que el sistema se encuentre funcional. Los requerimientos no funcionales son propiedades o cualidades que el producto debe poseer. Algunas de estas características son las que le conceden al producto las propiedades deseadas en él, estas se mencionan a continuación.

2.3.1 *SOFTWARE*.

RNF_1. El sistema deberá instalarse sobre Nova Ligero o cualquier distribución de Nova.

RNF_2. Brindar integración con el Sistema de Compilación Distribuida del departamento Nova (SCDN).

2.4 DIAGRAMA DE CASOS DE USO DEL SISTEMA.

En la figura 5 ubicada en anexo 1 del documento se muestran los casos de uso del sistema así como las relaciones existentes los diferentes casos de uso y los actores del sistema.

2.5 DESCRIPCIÓN DE LOS CASOS DE USOS DEL SISTEMA

Las tablas (desde la dos a la siete) que aparecen en el Anexo 1 del documento contienen la descripción de cada uno de los casos de uso por los cuales fue guiado el proceso de desarrollo.

2.6 Diagrama de Clases del Diseño.

Representa las clases que serán utilizadas dentro del sistema y las relaciones que existen entre ellas. Sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de convencimiento. Un diagrama de clases está compuesto por los siguientes elementos: Clase: atributos,

métodos y visibilidad; y Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

En la figura 6 se muestra el diagrama de clases que se definidas en la propuesta planteada.

La clase “Centro_de_control” se diseñó como una clase controladora encargada de gestionar y contener la clase “Hergire_Class” y la clase “Mantenedor”.

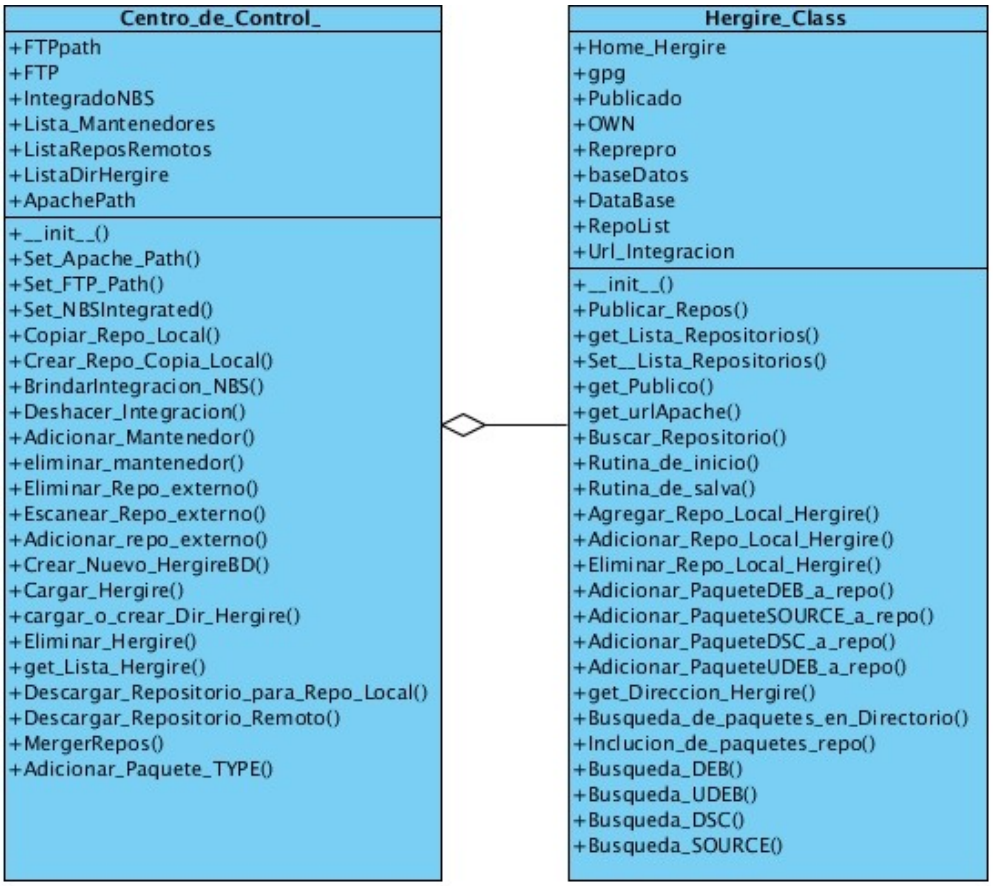

Figura 8: Las clases “Hergire_Class” son contenidas en el “Centro_de_Control_”

La clase Hergire_Class de tipo controladora es la encargada de la creación de nuevos repositorios así como de manejar los ya existentes. Controla también la base de datos de cada directorio mediante la clase DataBase y maneja mediante la clase RepoBinding

y GnuPG_Handler que hacen efectivos los cambios de los repositorios de la solución en el SO.

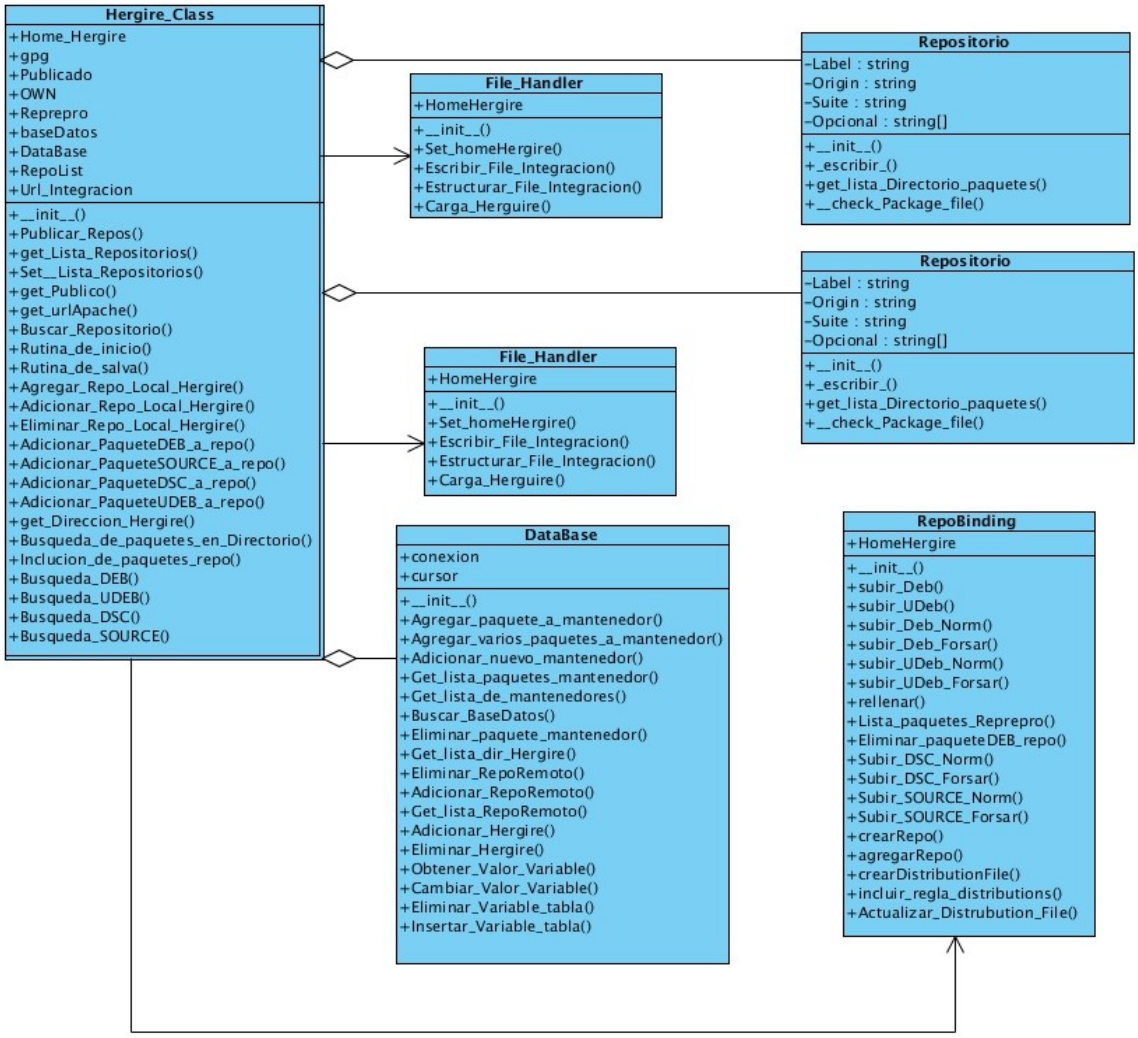

Figura 9: Las clases "Hergire_Class" es la encargada de gestionar los repositorios locales.

La clase Mantenedor es de tipo entidad y contiene los datos de cada mantenedor que se cree en el sistema. RepoBinding y GnuPG_Handler no son más que interfaces para interactuar con Reprepro y GnuPG respectivamente. File_Handler es una biblioteca implementada para el manejo de algunos los ficheros de texto.

2.7 Diagrama de vista lógica.

Representa un subconjunto del artefacto Modelo de Diseño. Este describe las clases más importantes, su organización en paquetes y subsistemas, éstos a su vez en capas siempre que sea posible.

En la figura 7 ubicada en el anexo 4 del documento, se muestran las clases organizadas, haciendo uso del Modelo Vista Controlador, el cual es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocio de la lógica de diseño. En el paquete controlado se encuentran las clases donde realiza toda lógica del negocio, en el paquete Modelo se encuentra las clases que van a tener implementado todo el acceso a datos.

2.8 Diagrama de Interacción.

Un diagrama de interacción consiste en un conjunto de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos. Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema. Ambos diagramas (secuencia y colaboración) son semánticamente equivalentes. Se puede pasar de uno a otro sin pérdida de información y se utilizan para modelar los aspectos dinámicos de un sistema.

2.8.1 Diagrama de colaboración.

Es esencialmente un diagrama que muestra interacciones organizadas alrededor de los roles. A diferencia de los diagramas de secuencia, los diagramas de colaboración, también llamados diagramas de comunicación, muestran explícitamente las relaciones de los roles. Por otra parte, un diagrama de comunicación no muestra el tiempo como una dimensión aparte, por lo que resulta necesario etiquetar con números de secuencia tanto la secuencia de mensajes como los hilos concurrentes.

En el anexo 8 del presente trabajo se pueden encontrar los diagramas de colaboración pertenecientes a la implementación algunos requisitos de la solución.

2.9 Patrones de diseño.

Describe un problema que ocurre varias veces y también describe el núcleo de la solución al problema, de forma que puede utilizarse en ilimitadas ocasiones sin tener que hacer dos veces lo mismo.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. En otras palabras, un patrón de diseño no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. [15.]

2.9.1 Patrones GRASP

GRASP es un acrónimo de *General Responsibility Assignment Software Patterns* (patrones generales de *software* para asignar responsabilidades). El nombre se eligió para sugerir la importancia de aprender estos principios para diseñar con éxito el *software* orientado a objetos.

Los patrones GRASP son utilizados para describir los principios fundamentales del diseño y la asignación de responsabilidades. [15.]

2.9.1.1 Experto en información

Un modelo de diseño podría definir cientos o miles de clases *software*, y una aplicación podría requerir que se realicen cientos o miles de responsabilidades. Durante el diseño de objetos, cuando se definen las interacciones entre los objetos, se toman decisiones sobre la asignación de responsabilidades a las clases. Haciendo un resumen de lo anteriormente planteado este patrón consiste en asignar la responsabilidad a la clase que tiene la información necesaria para realizarla.

En la solución se pone de manifiesto a en las clases del modelo de datos las cuales son encargadas de manejar subsistemas. Estas clases contienen toda la información y los métodos necesarios para realizar sus tareas como manejador del sistema en el cual esta especializado. Ejemplo: gpg, mantenedor_Class.

2.9.1.2 Creador

La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia, es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien, el diseño puede soportar un bajo acoplamiento, mayor claridad y reutilización. El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, una tarea muy común. La intención básica del patrón es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento.

Se pone en evidencia al crear clases Repositorios solo en las clases Hergire_Class ya que son las encargadas de contener y manejar los repositorios locales creados en el disco duro del ordenador.

2.9.1.3 Alta cohesión

La cohesión es la medida de la fuerza que une a las responsabilidades de una clase. Una clase con baja cohesión hace muchas cosas no relacionadas, o hace demasiado trabajo. Tales clases no son convenientes ya que son difíciles de mantener, de reutilizar y de entender. Una clase con alta cohesión mejora la claridad y la facilidad de su uso, su mantenimiento se simplifica y es fácil de reutilizar.

Se pone de manifiesto al cada módulo y clase con que cuenta la solución realiza un solo grupo de tareas en las cuales esta especializada. Lo que simplifica su entendimiento y mantenimiento. Ejemplo: gpg, RepoBinding.

2.9.1.4 Bajo acoplamiento

El acoplamiento es una medida de la fuerza con que un elemento está conectado a otro, un elemento con bajo acoplamiento no depende demasiado de otros elementos. Una clase con alto acoplamiento confía en muchas otras clases, tales clases podrían no ser deseables ya que son muy complicadas de entender, son difíciles de reutilizar y los cambios realizados en las clases relacionadas fuerzan cambios locales.

Es fácilmente probable en la herramienta ya que cada módulo funciona independientemente. Ejemplo: repo_scan

2.10 Conclusiones del capítulo

A lo largo de este capítulo se realizó el diseño de la aplicación. Se definieron los requisitos funcionales y no funcionales para el funcionamiento de la aplicación. Se modeló el Diagrama de Casos de Usos del Sistema, el Diagrama de Clases del Diseño y los Diagramas de interacción (comunicación) para una mejor comprensión de las funcionalidades del sistema. Se usara el patrón arquitectónico Modelo Vista Controlador (MVC) ya permite separar las vistas de la lógica del negocio y del modelo de datos.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA.

En el presente capítulo se realizará el diagrama de despliegue y el diagrama de componentes. Del diagrama de despliegue se observan sus relaciones y la descripción de los principales componentes que lo conforman. Además se realizó la implementación y la validación de la herramienta para la gestión de repositorios de paquetes de *software*.

3.1 DIAGRAMA DE DESPLIEGUE

El diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado *(UML por sus siglas en ingles)* que se utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes. [19.]

Los elementos usados por este tipo de diagrama son:

- **Nodos:** elementos de procesamiento con al menos un procesador, memoria, y posiblemente otros dispositivos.
- **Dispositivos:** nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
- **Conectores:** expresan el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.

En la figura 11 ubicada en los anexos se muestra el diagrama de despliegue del sistema. En el cuál se aprecia:

- **El nodo PC Administrador:** representa la máquina cliente asignada al administrador del sistema a cargo de los repositorios que se gestionen en el sistema. Este nodo estará conectado a un servidor Hergire.
- **El nodo servidor Hergire:** representa el servidor de la herramienta la cual cuenta con los repositorios que son gestionados por el sistema.

- **El nodo servidor del sistema de construcción de personalizaciones de Nova (SCPN):** representa el servidor de la aplicación SCPN, la cual usa la herramienta Hergire para obtener los repositorios necesarios para su gestión.
- **El nodo servidor sistema de compilación de Nova (SCN):** representa el servidor de la aplicación SCN, la cual usa la herramienta Hergire para obtener los repositorios necesarios para su gestión.

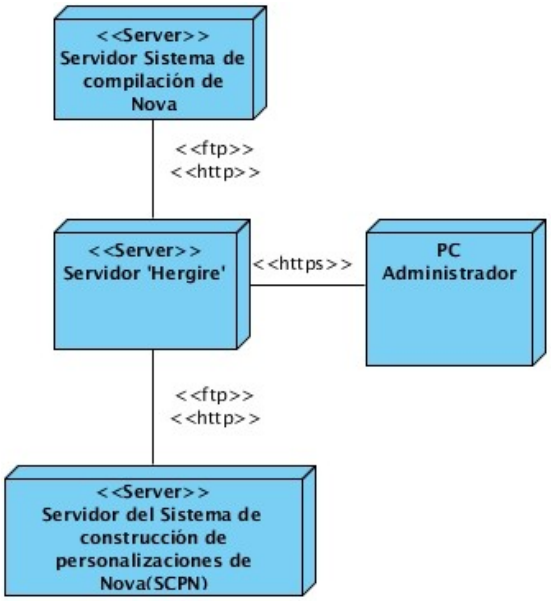

Figura 10: Diagrama de despliegue

3.2 Diagrama de Componentes

El diagrama de componentes es un tipo de diagrama que muestra las organizaciones y dependencias lógicas entre los componentes del *software* (ya sean código fuente, binarios o ejecutables). [20.]

Los elementos de modelado dentro de un diagrama de componentes son:

- Componentes: incluyen bibliotecas, tablas, archivos, ejecutables y documentos que formen parte del sistema.
- Paquetes: representan subsistemas.

En la figura 8 se muestra el diagrama de componentes del sistema propuesto. Dónde:

- El componente ControlCenter.py representa la el elemento central de la arquitectura. Contiene las clases y métodos que realizan la mayor cantidad de requisitos el sistema.
- El componente Hergire.py: Contiene la clase y los métodos que gestionan el manejo de los repositorios. Representa un directorio de configuración de repositorios.
- El componente Repo_Finder : representa un elemento que facilita la interacción con las funciones del protocolo http y permite escanear y agregar nuevos repositorios remotos a la base de datos.
- El componente mantenedorClass.py: representa la entidad mantenedor que gestiona el sistema.
- El componente gpg: le permite a la herramienta interactuar con el sistema de cifrado y autenticación GnuPG.
- El componente DataBase_Handler: Representa una clase que intermedia la comunicación con la base de datos Sqlite.
- El componente HergireDB.db representa la base de datos creada y gestionada con SQLite en su versión 3.

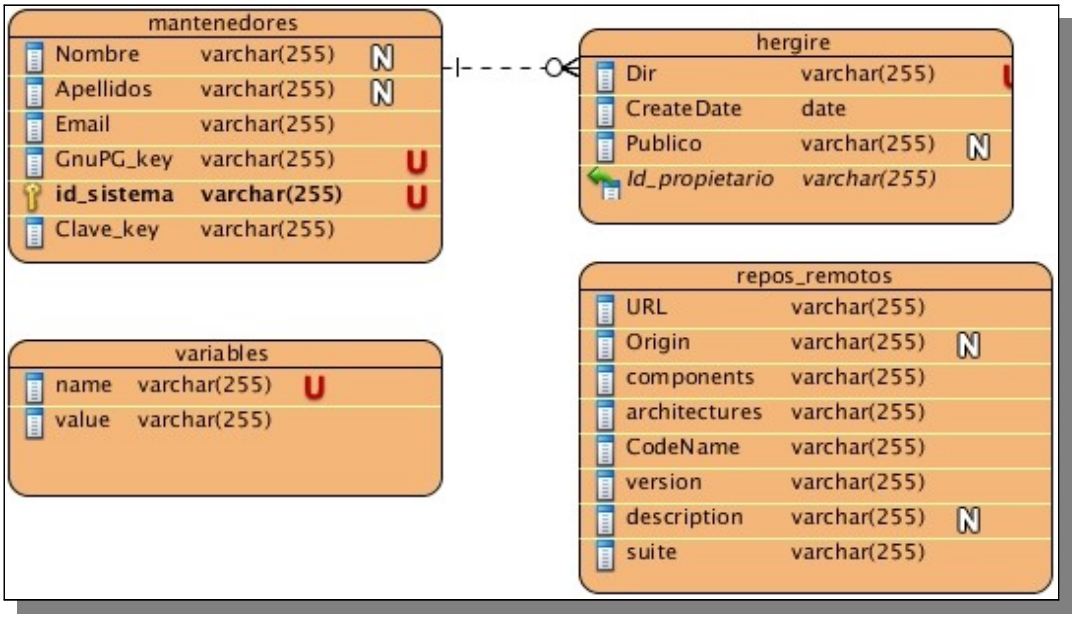

Figura 11: Descripción de la base de datos HergireDB.db

- **mantenedores:** Contiene los datos de los mantenedores registrados en el sistema.
- **variables:** Contiene los valores de las variables que necesita la herramienta para su correcto funcionamiento así como los datos introducidos por el administrador en la configuración establecida por él mismo.
- **hergire:** Datos de los directorios para la gestión de repositorios.
- **repos_remotos:** Datos de los repositorios externos al servidor.

- El componente FTP: contiene las rutinas posibles a realizar en el directorio compartido en el servidor FTP.
- El componente NBS: contiene información de repositorios que se le brinda al sistema de compilación de Nova.
- El componente FTP_SCPN: contiene información de repositorios que se le brinda a la herramienta SCPN.
- El componente **conf/distributions**: constituye un fichero de configuración de Reprepro utilizado para contener las configuraciones de los repositorios ubicados a nivel del directorio *conf*.

distributions

```
1  Origin: Debian
2  Label: Debian
3  Suite: stable
4  Codename: etch
5  Version: 4.0
6  Architectures: i386
7  Components: main
8  Description: Debian 4.0 etch + security updates
9  Update: - debian security
10 Log: logfile
11
12 Origin: nova2011
13 Label: nova
14 Suite: legacy
15 Codename: nova2011
16 Version: 3.0
17 Architectures: i386 amd64 sparc source
18 Components: main contrib universe multiverse
19 Description: Paquetes creados o backportados por mantenedor para nova
20
21 Origin: precise-security
22 Label:
23 Suite: precise-security
24 Codename: precise
25 Version: 12.04
26 Architectures: powerpc amd64 armel armhf i386
27 Components: multiverse main restricted universe
28 Description: Ubuntu Precise Security
```

Figura 12: Fichero de configuración de Reprepro conf/distributions.

- **Origin:** Origen de datos.
- **Label:** No es de mucha utilidad y se puede encontrar vacío o ausente. *Software* como el Synaptic[19] lo usan para ubicar el paquetes según alguna clasificación. Normalmente tiene correspondencia con los componentes del repositorio, main, contrib, etc...
- **Suite:** especifica la rama o ramas que va a tener el repositorio. En caso de que un paquete que se desea incluir tenga especificada la suite a la cual tiene definido ubicarse y en la descripción del repositorio anfitrión no este especificada la misma suite, se procederá a forzar la importación colocando el paquete en la rama predefinida del repositorio.
- **Codename:** Nombre o código de la distribución a la que se construye el repositorio. El valor de este campo es necesario y no se debe repetir en un mismo fichero de configuración.

19 **Synaptic**. Es un programa informático que es una interfaz gráfica GTK+ de APT, para el sistema de gestión de paquetes de Debian GNU/Linux. [22.]

- **Version:** Junto al Codename constituye la identificación de un repositorio en el directorio donde se encuentren.
- **Architectures**: Lista de arquitecturas de hardware para la cual estarán compilados los paquetes que contenga dicho repositorio.
- **Components:** Lista de componentes en los cuales se va a organizar el repositorio en cuestión.
- **Description:** Breve descripción o notas del repositorio.

• El componente **conf/updates**: constituye un fichero de configuración de Reprepro utilizado para contener las reglas de actualización y descarga de paquetes para los repositorios ubicados a nivel del directorio “conf”.

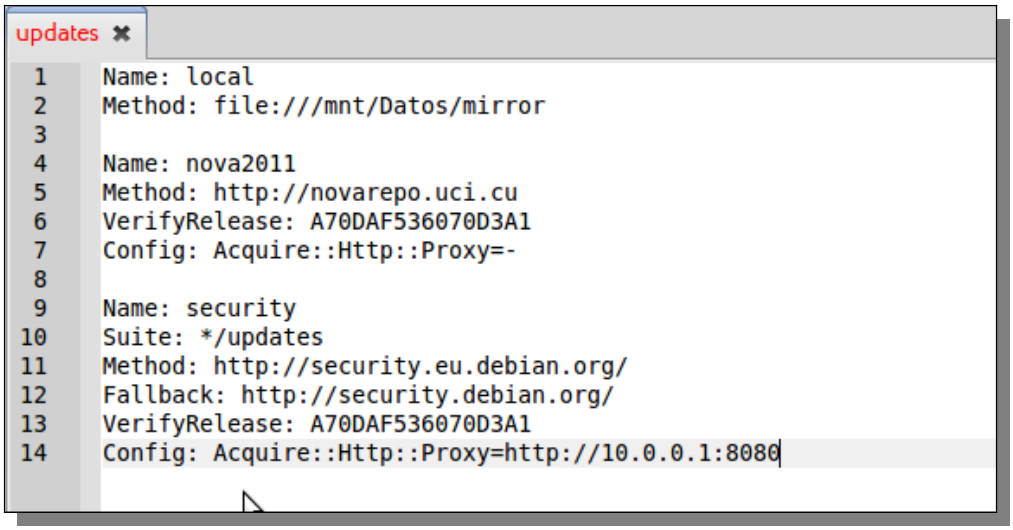

Figura 13: Fichero de configuración de Reprepro conf/updates.

- **Name:** Nombre de la regla especificada.
- **Method:** URL del origen de las actualizaciones.
- **VerifyRelease:** Id público de la llave usada para descargar paquetes del repositorio. Apt[20] debe contener la llave pública especificada.

20 **Advanced Packaging Tool** (Herramienta Avanzada de Empaquetado), abreviado **APT**, es un sistema de gestión de paquetes creado por el proyecto Debian. APT simplifica en gran medida la instalación y eliminación de programas en los sistemas GNU/Linux.

- **Config:** Especificación del proxy[21] y el protocolo a usar para la descarga.

3.3 Validación funcional

La calidad del *software* debe implementarse a lo largo de todo el ciclo de vida del producto, debe correr paralela desde la planificación del producto hasta la fase de producción de es te como actividad de protección. Uno de los aspecto a considerar en el control de la calidad del s*oftware* son las Pruebas de *Software*.

3.3.1 Pruebas de *software*

La prueba de *software* es un concepto que a menudo, es conocido como verificación y validación. Integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del *software*. Entre algunas de las técnicas que se realizan para el proceso de prueba se encuentran las técnicas de caja negra y de caja blanca.

3.3.1.1 Pruebas de caja negra.

Cuando se habla de un *software*, la prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del mismo. Los métodos de prueba de la caja negra se centran en los requisitos funcionales del mismo e intentan encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación.

3.3.1.1.1 Pruebas de Validación.

La validación del *software* se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Un plan de prueba traza la clase de

21 **servidor proxy**, que sirve para permitir el acceso a Internet

pruebas que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos. Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se alcanzan todos los requisitos de rendimiento, que la documentación es correcta e inteligible y que se alcanzan otros requisitos (por ejemplo, portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento) [21.].

Una vez que se procede cada caso de prueba de validación, puede darse una de las dos condiciones siguientes:

1) Las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptables.
2) Se descubre una desviación de las especificaciones y se crea una lista de deficiencias. Las desviaciones o errores descubiertos en esta fase del proyecto raramente se pueden corregir antes de la terminación planificada. generalmente es necesario negociar con el cliente un método para resolver las deficiencias [21.].

3.3.1.2 Pruebas de caja Blanca.

La prueba de caja blanca, denominada a veces prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de *software* puede obtener casos de prueba que:

1. Garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.
2. Ejerciten todas las decisiones lógicas en sus vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez [21.].

3.3.1.2.1 Pruebas unitarias.

La implementación de pruebas unitarias permite probar el correcto funcionamiento de cada uno de los módulos por separado. Se deben escribir casos de prueba para cada

función de forma que cada uno sea independiente del resto. En el desarrollo se utiliza el módulo *unittest* de Python tratando de cubrir la mayor cantidad de código posible [4.].

Mediante los diseños de casos de prueba basados en requisitos, realizados a la interfaz de la herramienta y contenidos en el Anexo 7 del presente documento. Se detectaron cinco no conformidades pertenecientes a las validaciones de los datos proporcionados por el usuario y otras 2 correspondientes a los proceso de eliminación de paquetes. Estas no conformidades fueron resueltas correctamente.

3.4 Conclusiones del capítulo

En el presente capítulo se realizó la implementación del sistema, para ello se diseñó los principales componentes y sus relaciones, los cuales fueron mostrados de forma sencilla mediante el diagrama de componentes, donde se muestran los ficheros de configuraciones más importantes y la descripción de la base de datos. Además se hizo la modelación del diagrama de despliegue. Para realizar la validación del sistema se decidió utilizar la técnica de caja negra la cual fue aplicada en el diseño de pruebas.

CONCLUSIONES

Como resultado del estudio anterior ha sido posible la construcción de un sistema que da respuesta al problema científico planteado en el capítulo introductorio del mismo, cumpliéndose el objetivo general definido al inicio de la investigación, entre los principales aspectos se destacan:

1. Se identificaron los principales recursos y herramientas a estudiar.
2. Se analizó y diseño una herramienta para la gestión de repositorios de *software*.
3. Se obtuvo una gestión sencilla y eficaz que simplifica y organiza el proceso de gestión de repositorios de paquetes de *software*.
4. Se logró automatizar procesos que anteriormente no se realizaban.

RECOMENDACIONES

En aras de mejorar algunos aspectos del producto Hergire, se deben tener en cuenta las siguientes recomendaciones para posteriores desarrollos de dicha herramienta:

1. La comunicación entre esta y las demás herramientas de la plataforma de desarrollo de Nova debería ser implementada con RSS. Mejorando su seguridad y optimizando los recursos del servidor.
2. Implementar un sistema de salvas sustituyendo las bases de datos empleadas por Reprepro en los directorios gestionados. Esto para en caso de que la base de datos se corrompa, se tenga un respaldo de todos los datos.
3. Implementar un módulo que permita a los mantenedores interactuar con los repositorios de paquetes e incorporar sus paquetes personalizados.

Referencia Bibliográfica

1 Bernhard R. Link. REPREPRO. 2011. [2 December 2011]. Disponible en la Web: <http://mirrorer.alioth.debian.org/reprepro.1.html>.

2 Ubuntu.com. Debmirror - Community Ubuntu Documentation. Enero 2010. [4 December 2011]. Disponible en la Web: <https://help.ubuntu.com/community/Debmirror>.

3 wiki. GNU Privacy Guard - Wikipedia, la enciclopedia libre. October 2011b. [4 December 2011]. Disponible en la Web: <http://es.wikipedia.org/wiki/GNU_Privacy_Guard>.

4 wiki. Perl - Wikipedia, la enciclopedia libre. November 2011c. [5 December 2011]. Disponible en la Web: <http://es.wikipedia.org/wiki/Perl>.

5 wiki. Pretty Good Privacy - Wikipedia, la enciclopedia libre. 27 2011d. [4 December 2011]. Disponible en la Web: <http://es.wikipedia.org/wiki/PGP>.

4. Yunier Soler Franco. Serere 2.0. Instalador del sistema operativo GNU/Linux Nova. May 2010.

5. Daniel Hernandez Bahr. Sistema para la generación de paquetes binarios fragmentados a partir del código fuente. May 2010.

6. GnuPG users. GnuPG Frequently Asked Questions. June 2011. [3 December 2011]. Disponible en la Web: <http://www.gnupg.org/faq/GnuPG-FAQ.html#what-is-gnupg>.

7. geany.org. Geany : About. 2011. [10 December 2011]. Disponible en la Web: <http://www.geany.org/Main/About>.

8. Mónica Ma Albo Castro. Sistema para automatizar la generación de paquetes binarios de la distribución Nova. June 2008. [13 December 2011].

9. http://dev.mysql.com/. MySQL :: MySQL 5.0 Reference Manual :: 1.4.2 Las principales características de MySQL. *Las principales características de MySQL* February 2012. [2 March 2012]. Disponible en la Web: <http://dev.mysql.com/doc/refman/5.0/es/features.html>.

10. wiki. PostgreSQL - Wikipedia, la enciclopedia libre. *PostgreSQL* February 2012. [2 March 2012]. Disponible en la Web: <http://es.wikipedia.org/wiki/PostgreSQL>.

11. EcuRed. SQLite - EcuRed. February 2012. [2 March 2012]. Disponible en la Web: <http://www.ecured.cu/index.php/SQLite>.

12. *Metodologías de desarrollo de software* May 2012. [5 March 2012]. Disponible en la Web: <http://www.rhernando.net/modules/tutorials/doc/ing/met_soft.html>.

13. Metodologias de desarrollo *software*. [5 March 2012]. Disponible en la Web: <http://es.scribd.com/Samirak/d/2050925-metodologias-de-desarrollo-software>.

14. Cbasqa. Proceso de Desarrollo OpenUP « CBASQA – Desarrollo de *Software*, SQA, Testing, Servicios Informáticos, Project Management. [5 March 2012]. Disponible en la Web: <http://cbasqa.wordpress.com/2008/09/02/proceso-de-desarrollo-openup/>.

15. Craig Larman. *UML y Patrones. Indroducción al análisis y diseño orientado a objetos.* 1era edición, 1999 [Mexico]: PRENTICE HALL.

16. wiki. RPM Package Manager - Wikipedia, la enciclopedia libre. August 2012. [8 March 2012]. Disponible en la Web: <http://es.wikipedia.org/wiki/RPM_Package_Manager>.

17. EcuRed. Red Hat Linux - EcuRed. August 2012. [8 March 2012]. Disponible en la Web: <http://www.ecured.cu/index.php/Red_Hat_Linux>.

18. Wiki. File Transfer Protocol - Wikipedia, la enciclopedia libre. [8 March 2012]. Disponible en la Web: <http://es.wikipedia.org/wiki/File_Transfer_Protocol>.

19. Sparxsystems. [En línea] [Citado el: 15 de Febrero de 2011.] <http://www.sparxsystems.com.ar/download/ayuda/index.html?deploymentdiagram.html>.

20. Departamento de Sistemas Informáticos. [En línea] [Citado el: 18 de Febrero de 2011.] <http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf>.

21. Roger S. Pressman. *Ingeniería de software. Un enfoque práctico.* 5ta edición. Mc Graw Hill.

22. www.ecured.cu. Synaptic - EcuRed. [24 May 2012]. Disponible en la Web: <http://www.ecured.cu/index.php/Synaptic>.

23. Anon. Lenguajes de programación: Python. [29 November 2011]. Disponible en la Web: <http://hosting.udlap.mx/profesores/miguela.mendez/alephzero/archivo/historico/az26/python.html>.

24. Melodysof . Melodysoft. LOS LENGUAJES DE PROGRAMACION. [1 December 2011]. Disponible en la Web: <http://boards5.melodysoft.com/2007AISC0107/los-lenguajes-de-programacion-15.html>.

25. Slideshare. Conceptos Basicos Programacion. [1 December 2011]. Disponible en la Web: <http://www.slideshare.net/mandre55/conceptos-basicos-programacion>.

Glosario de términos

Ambientes Integrados de Desarrollo (IDE): Aplicación que provee facilidades para el desarrollo de *software*, por parte de los programadores.

API (*Aplication programming interface*): conjunto de funciones utilizados para comunicarse con otros programas.

Código fuente: Instrucciones y expresiones de un programa, escritas por el programador en un lenguaje determinado. No es ejecutable directamente por la computadora. Puede ser escrito en un editor de texto y guardado en un archivo que luego hay que convertir a código máquina para que la computadora “lo entienda”. Cuando el código fuente de un programa es de libre acceso se dice que es código abierto.

Comunidad: Grupo de individuos con características comunes que comparten un espacio común.

Comunidad de desarrollo: Comunidad de personas que trabajan en alas de desarrollar una y lograr un objetivo de interés común.

Depurador: (en inglés, debugger), es un programa usado para probar y depurar (eliminar los errores) de otros programas (el programa "objetivo"). El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente algo más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador.

Fichero de configuración: Fichero usado para guardar la configuración inicial de algunos programas de computadoras.

GNU: Es el nombre del proyecto creado en 1984 por Richard Stallman para crear un sistema operativo totalmente libre, es un acrónimo recursivo que significa GNU no es Unix .

Hardware: Término general para los artefactos físicos de una tecnología. Puede aplicarse a los componentes físicos de un sistema de cómputo.

MD5: Algoritmo de reducción criptográfico de 128 bits altamente usado.

PGP (*Pretty Good Privacy*): Programa informático que provee privacidad criptográfica y autenticación. A menudo se utiliza para firmar, cifrar y descifrar correos electrónicos para aumentar la seguridad de las comunicaciones por esta vía. Fue creado por Philip Zimmermann en 1991.

Proxy. En el contexto de las red informática, el término hace referencia a un programa o dispositivo que realiza una acción en representación de otro. Su finalidad más habitual es la de **servidor proxy**, que sirve para permitir el acceso a Internet a todos los equipos de una organización cuando sólo se puede disponer de un único equipo conectado, esto es, una única dirección IP.

***Software*:** Término general fundamentalmente utilizado para datos almacenados digitalmente, tales como programas de computadoras y otros tipos de información leída y escrita por computadoras. El término fue acuñado de manera que contrastara con el término hardware más antiguo. En contraste con el hardware el *software* es intangible, significando que "no puede ser tocado". En ocasiones el término se utiliza más estrechamente para referirse únicamente a aplicaciones de *software*.

RFC-4880: Estándar que establece las características de los sistemas OpenGPG. Define los métodos necesarios para realizar las operaciones básicas necesarias para la comunicación. Contiene consejos y buenas prácticas para evitar comprometer la seguridad de las aplicaciones.

XML: Por sus siglas en inglés de ***Ex****tensible* ***M****arkup* ***L****anguage* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el *World Wide Web Consortium* (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML).

Anexos

Anexo 1 Descripción de casos de uso.

Tabla gestionar repositorio de paquetes de *software* .

Objetivo	*Gestionar un repositorio binario*
Actores	*Administrador*
Resumen	*El administrador de un repositorio de paquetes de software binario gestiona el mismo.*
Complejidad	*Media*
Prioridad	*criticof*
Precondiciones	• *El administrador debe poseer un usuario creado en el sistema del servidor.* • *El mantenedor debe haber marcado y acceder debidamente a un directorio en el sistema donde esté creado correctamente la estructura de archivos del sistema.* • *Debe estar inicializado el directorio seleccionado, cargando la base de datos e interpretando los ficheros de configuración de Reprepro.* • *Mostrada una lista de los repositorios existentes en dicho directorio además de sus respectivos datos.*
Un Postcondiciones	

Flujo de eventos		
Flujo básico: Crear repositorio de paquetes de *software*		
	Actor	**Sistema**
1.	*Inicia el caso de uso.*	1. El sistema solicita los datos del repositorio a crear.

2.	*Introduce los datos solicitados*	1. Crea un objeto repositorio con los datos proporcionados y verifica si existe ya alguno con idéntico nombre clave y versión. 2. De ser verdadera la comprobación; Ver flujo alterno No.1 3. En caso negativo se agregan los datos del nuevo repositorio a los ficheros de configuración del Reprepro. 4. En la interfaz se actualizan los datos del directorio Hergire que ha sido modificado.
3.		• *Fin del caso de uso*
Flujos alternos		
Nº 1. Repositorio Existente.		
	Actor	**Sistema**
1.		7. Lanza un error de repositorio existente.
2.		Fin de flujo alterno.
Sección 1: "Eliminar repositorio de paquetes de *software* "		
Flujo básico: "Eliminar repositorio de paquetes de *software* binario"		
	Actor	**Sistema**
1.	Inicializa el caso de uso	1. Se extrae el repositorio seleccionado de la lista de repositorios de Hergire. 2. Busca en el fichero de configuración de Reprepro y elimina las líneas del fichero que correspondan con el repositorio eliminado. 3. En la interfaz se actualizan los datos del directorio Hergire que ha sido modificado.
2.		1. Fin del caso de uso

Sección 3: “Actualizar repositorio de paquetes de *software* ”		
Flujo básico “ Actualizar repositorio de paquetes de *software* ”		
1.	1. Inicia el caso de uso.	1. Ejecuta el comando correspondiente para el Reprepro. 2. Actualiza visual.
2.		1. Fin del caso de uso
Relaciones	**CU Incluidos**	
	CU Extendidos	*Firmar repositorio para Caso de uso Crear repositorio binario*
Requisitos no funcionales	• *Sistema operativo Nova.* • *Reprepro.* • *GnuPG.*	
Asuntos pendientes		

Tabla 2. Gestionar repositorio de paquetes de *software* .

TABLA GESTIONAR LLAVE DE MANTENEDOR.

Objetivo	*Manejar las llaves generadas por GPG para realizar las firmas de paquetes.*
Actores	
Resumen	*Crear y eliminar llaves gpg para firmas*
Complejidad	*Media*
Prioridad	*critico*
Precondiciones	• *GPG de debe encontrar instalado en el sistema*
Postcondiciones	
Flujo de eventos	
Flujo básico “Crear llave de mantenedor”	

	Actor	**Sistema**
1.		Mediante la interacción con la aplicación GnuPG, se crea una llave con los datos proporcionados para la creación de un mantenedor de paquetes.
2.		*Se almacena en la base de datos los datos del mantenedor y entre los datos, el nombre de la llave generada.*
3.		*Fin del caso de uso*
Sección 1: "Eliminar llave de mantenedor de paquetes"		
Flujo básico "Eliminar llave de mantenedor de paquetes"		
	Actor	**Sistema**
1.		1. Mediante la interacción con la aplicación GnuPG, se elimina la llave con los datos proporcionados para la creación de un mantenedor de paquetes.
2.		Fin del caso de uso
Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos no funcionales	*GnuPG.* *SQLite3*	
Asuntos pendientes		

Tabla 3. Gestionar llave de mantenedor

TABLA GESTIONAR MANTENEDOR DE PAQUETES.

Objetivo	*Manejar los usuarios que cuentan con llaves GPG y son capaces de firmar paquetes.*
Actores	*administrador*
Resumen	*Crear y eliminar llaves mantenedores de paquetes.*
Complejidad	*Media*

Prioridad	*critico*	
Precondiciones	• *El administrador que inicializa debe tener un usuario creado en el sistema del servidor.* • *GPG debe estar instalado en el sistema.*	
Postcondiciones		
Flujo de eventos		
Flujo básico “Crear mantenedor de paquetes”		
	Actor	**Sistema**
1.	*Inicializa caso de uso.*	1. Solicita los datos del nuevo mantenedor.
2.	Introduce los datos solicitados.	1. Crea una llave GPG para el mantenedor. 2. Guarda los datos del mantenedor en la Base de datos. 3. Agrega el nuevo mantenedor a la lista. *4. Actualiza visual.*
		Fin del caso de uso
Sección 1: “Eliminar mantenedor de paquetes”		
Flujo básico “Eliminar mantenedor de paquetes”		
	Actor	**Sistema**
1.	Inicializa caso de uso	1. Extrae mantenedor de la lista. 2. Eliminar mantenedor de la Base de datos. 3. Elimina llave GPG del sistema. 4. Actualiza visual
2.		*Fin del caso de uso*
Relaciones	**CU Incluidos**	

	CU Extendidos	*CU 4. Gestionar llave mantenedor para CU 5. Gestionar mantenedor de paquetes.*
Requisitos no funcionales	• *GnuPG.* • *SQLite3*	
Asuntos pendientes		

Tabla 4. Gestionar mantenedor de paquetes

TABLA CHEQUEAR ESTADO DE REPOSITORIO.

Objetivo	*Realizar informe sobre el estado de los paquetes de los repositorios*	
Actores	*administrador*	
Resumen	*Le brinda al administrador los datos del estado de los repositorios, además envía correos a los diferentes mantenedores de paquetes para notificarles en caso de alguna novedad con sus paquetes mantenidos.*	
Complejidad	*Media*	
Prioridad	*Alta*	
Precondiciones		
Un Postcondiciones		
Flujo de eventos		
Flujo básico: *Brindar informe de repositorio.*		
	Actor	**Sistema**
1.	*El mantenedor inicializa el caso de uso.*	1. Chequea las actualizaciones existentes para los paquetes del repositorio. 2. Chequea la integridad de los paquetes del repositorio. 3. Muestra un informe con los datos recogidos.
2.		*1. Fin del caso de uso*

Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos no funcionales	• *Sistema operativo Nova.* • *Reprepro.* • *SQLite3*	
Asuntos pendientes		

Tabla 5. Chequear estado de repositorio.

TABLA FIRMAR REPOSITORIO DE PAQUETES DE *SOFTWARE*.

Objetivo	*Firmar los índices de los repositorios creados por usuarios que posean llaves GPG.*
Actores	
Resumen	*Cuando se crea un repositorio de cualquier naturaleza, se necesita que el propietario firme el repositorio con su llave generada en GPG*
Complejidad	*Media*
Prioridad	*Alta*
Precondiciones	• *El mantenedor debe tener un usuario creado en el sistema del servidor..* • *Inicializado el directorio seleccionado, cargando la base de datos e interpretando los ficheros de configuración de Reprepro.* • *Mostrada una lista de los repositorios existentes en dicho directorio además de sus respectivos datos.*
Un Postcondiciones	
Flujo de eventos	
Flujo básico: "Firmar repositorio de paquetes de *software*".	

	Actor	Sistema
1.		1. Busca en la Base de datos El identificador de la llave GPG del mantenedor en el sistema. 2. Escribe la llave en los ficheros correspondientes del Reprepro
2.		• *Fin del caso de uso*
Relaciones	**CU Incluidos**	
	CU Extendidos	
Requisitos no funcionales	• *Sistema operativo Nova.* • *Reprepro.* • *GnuPG* • *SQLite3*	
Asuntos pendientes		

Tabla 6. Firmar repositorio de paquetes de *software*

TABLA MEZCLAR REPOSITORIO DE *SOFTWARE*.

Objetivo	*Mezclar dos repositorios de software cualquiera que sea su origen.*
Actores	*Administrador*
Resumen	*El administrador selecciona dos repositorios de software, selecciona un directorio destino y procede a mezclar.*
Complejidad	*Media*
Prioridad	*Alta*
Precondiciones	• *El administrador debe tener un usuario creado en el sistema del servidor.* • *Debe estar instalado el software de gestión de repositorios "Reprepro"*

<table>
<tr><td colspan="2">Un Postcondiciones</td><td colspan="2"></td></tr>
<tr><td colspan="4">Flujo de eventos</td></tr>
<tr><td colspan="4">Flujo básico: “Mezclar repositorio de paquetes de software</td></tr>
<tr><td></td><td colspan="2">Actor</td><td>Sistema</td></tr>
<tr><td>1.</td><td colspan="2">1. Inicializa el caso de uso.</td><td>1. Se muestra una lista donde se contienen los repositorios existentes en los datos de la herramienta.</td></tr>
<tr><td>2.</td><td colspan="2">1. Selecciona un repositorio y marca el botón “mezclar con”</td><td>1. Muestra una lista donde se contienen los repositorios existentes en los datos de la herramienta a excepción del ya seleccionado.</td></tr>
<tr><td>3.</td><td colspan="2">1. Selecciona la opción “mezclar en”</td><td>1. Se muestran todos los directorios disponibles donde ubicar el nuevo repositorio.</td></tr>
<tr><td></td><td colspan="2">1. Selecciona la opción “ Hacer mezclar”</td><td>1. Se crea un nuevo repositorio en el directorio especificado.
2. Se indizan y se descarga los paquetes del primer repositorio.
3. Se incluyen los paquetes indexados y descargados en el nuevo repositorio.
4. Se indizan y descarga los paquetes del segundo repositorio.
5. Se incluyen los paquetes indexados y descargados en el nuevo repositorio.</td></tr>
<tr><td></td><td colspan="2"></td><td>1. Fin del caso de uso</td></tr>
<tr><td colspan="2" rowspan="2">Relaciones</td><td>CU Incluidos</td><td></td></tr>
<tr><td>CU Extendidos</td><td>CU1,CU2,CU3</td></tr>
<tr><td colspan="2">Requisitos no funcionales</td><td colspan="2">• Sistema operativo Nova.
• Reprepro.
• GnuPG
• SQLite3</td></tr>
<tr><td colspan="2">Asuntos</td><td colspan="2"></td></tr>
</table>

pendientes	

Tabla 7. Mezclar repositorio de paquetes de *software*

ANEXO 2: DIAGRAMA DE CASOS DE USO.

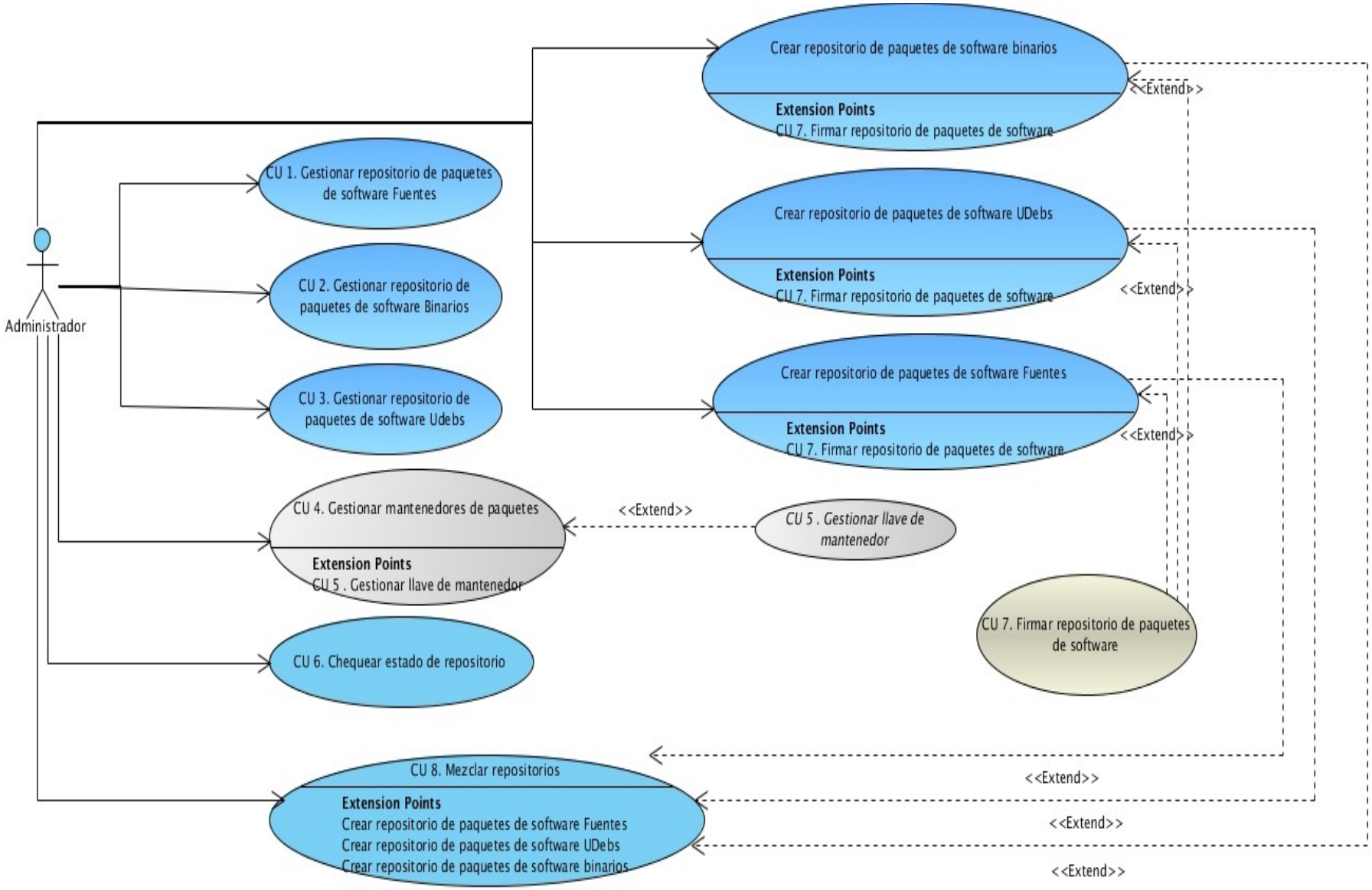

Figura 14. Diagrama de casos de uso.

ANEXO 3: DIAGRAMA DE CLASES DEL SISTEMA

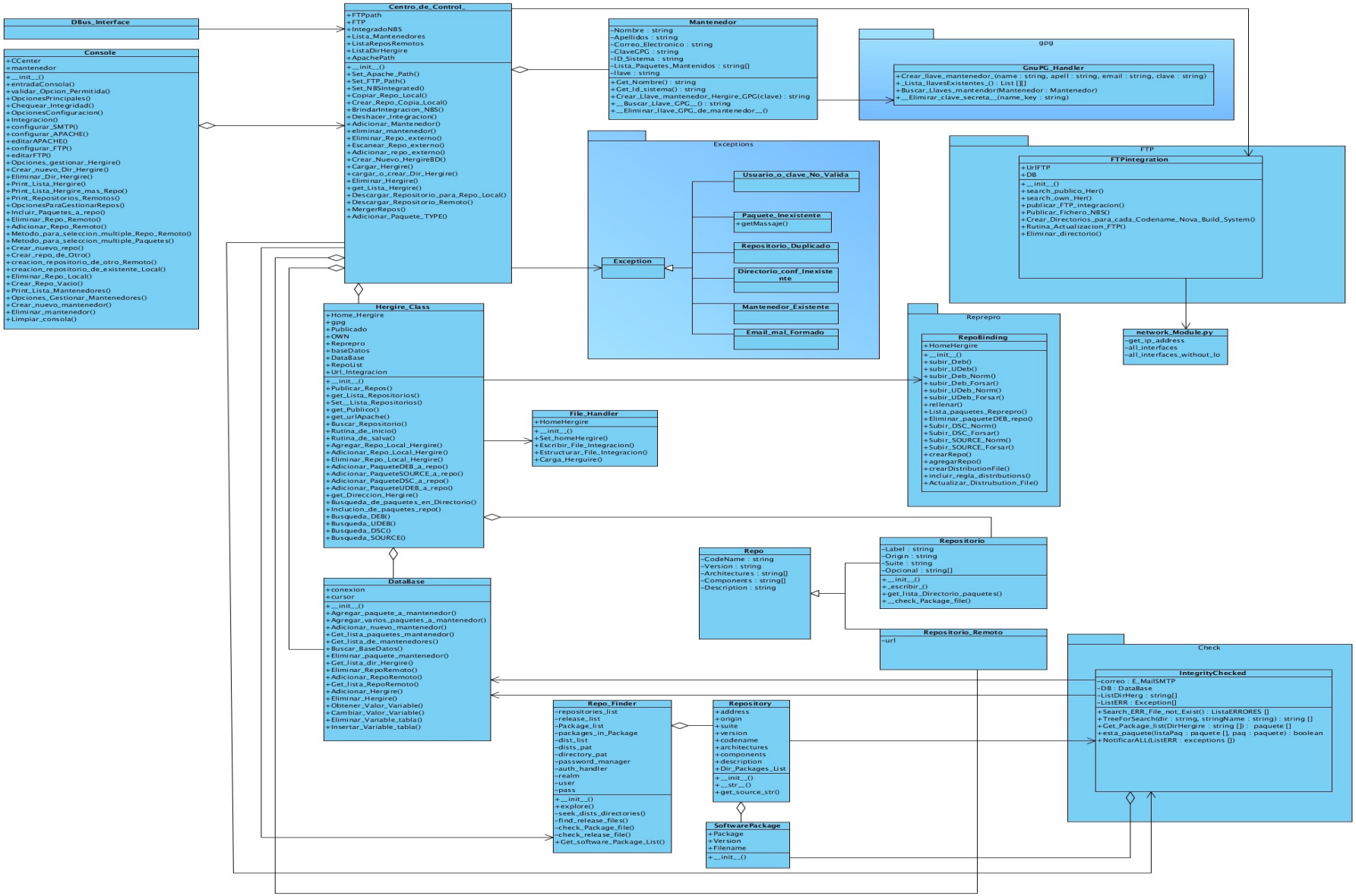

Figura 15. Diagrama de Clases del Sistema

ANEXO 4: DIAGRAMA DE VISTAS LÓGICAS.

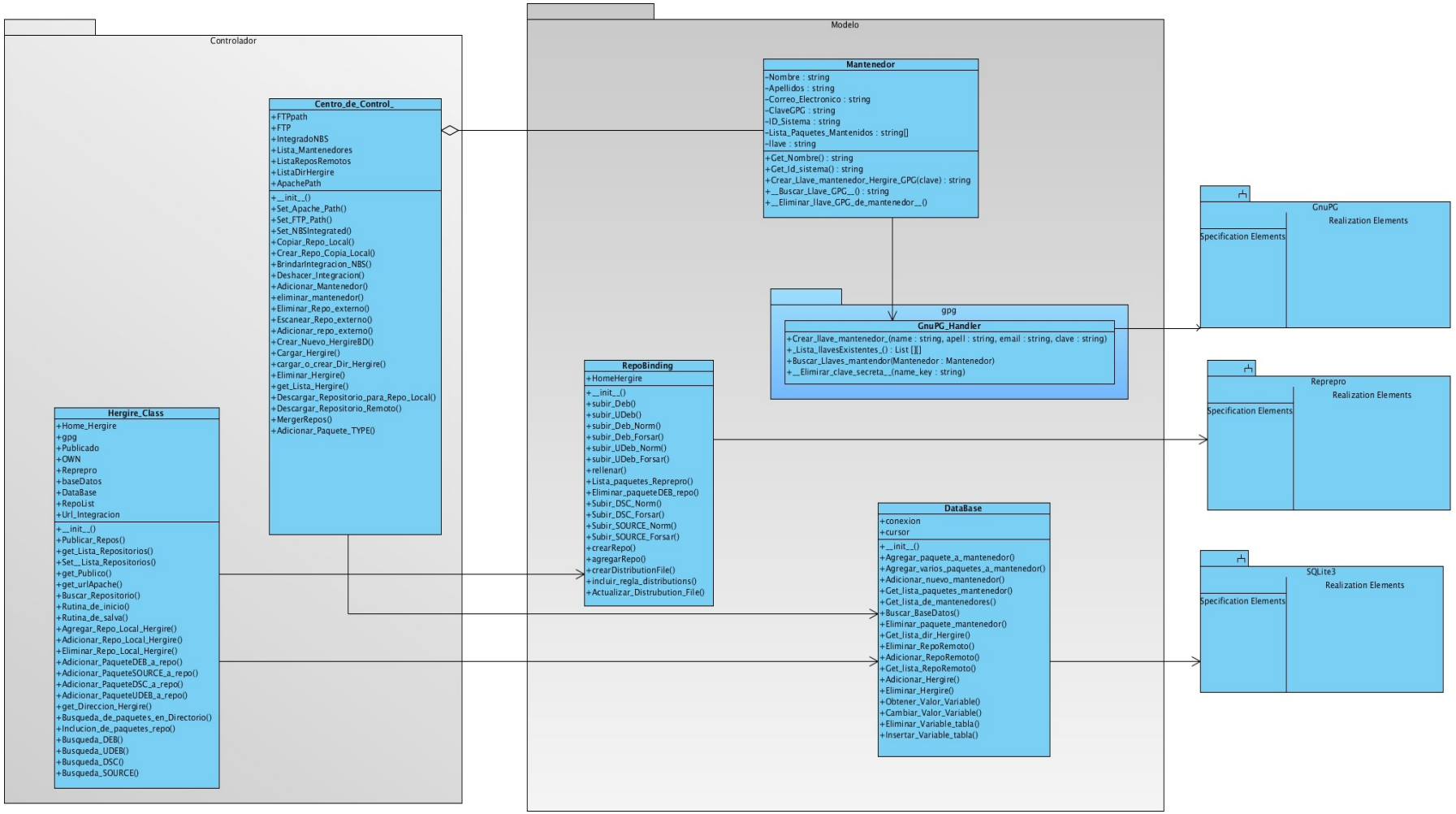

Figura 16: Diagrama de vistas lógicas.

ANEXO 5: DIAGRAMA DE COMPONENTES.

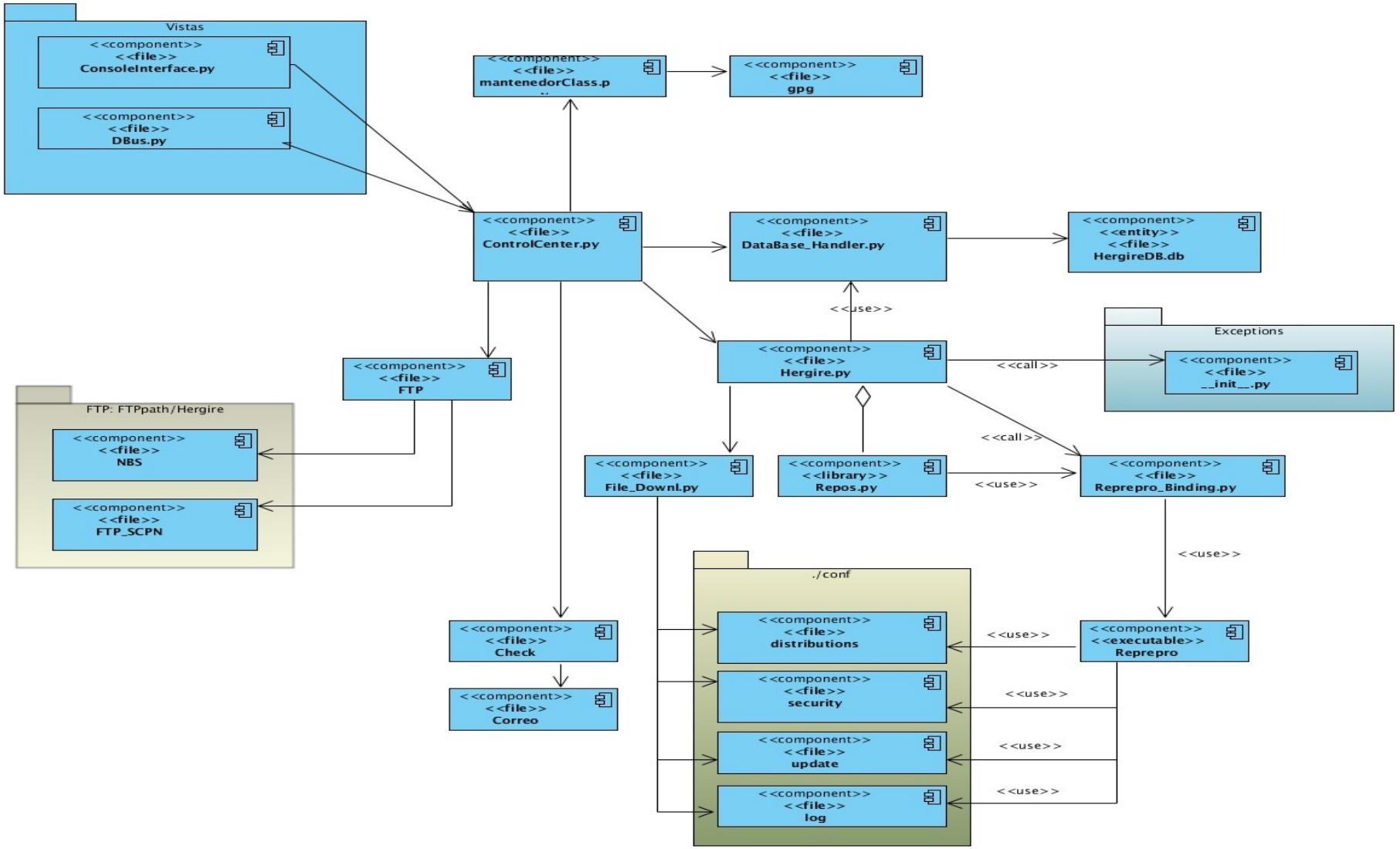

Figura 17. Diagrama de componentes

Anexo 6: Descripción de variables.

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Origin	campo de texto	si	Origen de los datos
2	Suite	campo de texto	si	A la hora de crear un paquete, en el changelog definimos una línea que contiene la versión del paquete y justo a continuación, la “suite” a la que corresponde. Generalmente Debian suele usar "*stable*", "*testing*" y "*unstable*", pero podemos poner cualquier cadena alfanumérica , siempre que pongamos la misma en Reprepro y en todos los paquetes.
3	Label	campo de texto	si	Es una etiqueta a tu elección, puedes poner cualquier cosa. No tiene mucha utilidad, pero en programas como synaptic cada categoría se divide en las etiquetas de los repositorios y si normalmente se corresponden con la rama del repositorio (*main*, *universe*, ...).
4	Codename	campo de texto	no	Nombre código de la distribución, Es buena idea usar el mismo de la distribución, para no liar a la gente, pero también podemos poner lo que queramos.
5	Version	campo de texto	si	Como el *codename*, pero con la versión numérica. Si el anterior es "dapper" aquí toca "6.06".
6	Architectures	campo de texto	no	Debemos listar cada una de las arquitecturas para las que incluimos soporte. Además de las arquitecturas de procesador (i386, ppc, sparc, ...) hay dos genéricas, "*source*" para paquetes de fuentes y "all" para paquetes independientes de la arquitectura, como scripts interpretados (scripts en lenguajes Bash, Python, ...).
7	Components	campo de texto	no	Es donde se especifican las ramas en que dividiremos el repositorio, por ejemplo "*main contrib universe multiverse*"
8	Description	campo de texto	si	Breve descripción del repositorio
10	Herg_origen	Lista	no	Representa el directorio que contiene el

				repositorio origen.
11	Repo_Origen	Lista	no	Representa el repositorio origen que contiene los paquetes a solicitados.
12	Herg_Destino	Lista	no	Representa el directorio destino donde se creara nuevo repositorio.
13	URL	Campo Selección	no	Representa la dirección donde se ubica el repositorio remoto en la red.
14	Nombre	campo de texto	no	nombre del mantenedor de paquetes
15	Apellidos	campo de texto	si	apellidos del mantenedor de paquetes
16	Email	campo de texto	no	E-mail del mantenedor de paquetes
17	ClaveGPG	campo de texto	no	contraseña del mantenedor
18	ID_sistema	campo de texto	no	ID o seudónimo del usuario en el sistema.
19	Repositorio	lista	no	Índice de objeto repositorio
20	Hergire_Class	lista	no	Índice de objeto Hergire
21	Repositorio_Remoto	lista	no	Índice de repositorio remoto

Tabla 15: Descripción de variables.

ANEXO 7: DISEÑOS DE CASOS DE PRUEBA.

7.1 CREAR REPOSITORIO.

SC <CP1. "Creación de repositorios">

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios Reprepro
El usuario debe estar bien identificado en el sistema.
Debe cargarse los datos de la base de datos pertenecientes a el directorio especificado y deben e interpretarse correctamente los ficheros de configuración de Reprepro ubicados en el directorio "/config" de los directorios Hergire existentes.
Debe mostrarse en la interfaz del visual una lista de repositorios existentes en el directorio especificado.
Debe estar instalado en el sistema operativo el servidor Apache, si se desea publicar el repositorio.

RF1, RF4, RF7,RF20 Crear repositorio de paquetes de software y Firmar repositorio.

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5	Variable 6	Variable 7	Variable 8	Respuesta del sistema	Flujo central
EC_1.1 Crear repositorio de paquetes de software vacío	Creación de un repositorio fuente vacío en un directorio especificado del servidor.	Origin	Suite	Label	Codename	Version	Architectures	Components	Description	0.2- El sistema muestra las opciones para la creación de un nuevo repositorio. 0.4- Consulta la base de datos buscando los directorios disponibles para la gestión de repositorio y los muestra. 0.6 – Solicita al usuario los daros para la creación del nuevo repositorio. "Origin","Label", "Suite", "Codename", "Version", "Architectures", "Components", "Description". 0.8- Escribe los datos en los ficheros de configuración correspondientes al directorio especificado junto con la especificación de la llave para firmar el repositorio en el fichero "distributions". 0.9 - Devuelve al usuario a la pantalla de gestión de repositorio.	0.1-Elige la opción de crear nuevo repositorio en la interfaz. 0.3 -Elige la opción "Crear repositorio vacio". 0.5 - Selecciona el directorio donde ubicar el nuevo repositorio. 0.7 - El usuario introduce los datos solicitados.
		>>cualquier_combinación_alfanumérica	>>cualquier_combinación_alfanumérica	>>cualquier_combinación_alfanumérica	>>nova2011 >>nova >>natty >>lucid-security >>nova-extremo >>Cualquier_combinación_alfanumérica	>>3 >>2da >>cualquier_combinación_alfanumérica	>>source >>source, i386,amd >>source,cualquier_otra_arquitecturas	>>main,contrib,universe,muntiverse >>main,non-free >>main	>>Cualquier_combinación_alfanumérica		
EC_1.2 Crear repositorio de paquetes de software a partir de una copia a un repositorio local ya existente	El usuario puede crear un repositorio local personalizado a partir de otro repositorio local existente.	Herg_origen	Repo_Origen	Herg_Destino	-	-	-	-	-	0.2 - El sistema muestra las opciones para la creación de un nuevo repositorio. 0.4- Muestra opciones de importación. 0.6 -El sistema consulta la base de datos buscando los directorios disponibles para la gestión de repositorio y los muestra. 0.8 -Muestra los repositorios existentes en el directorio seleccionado. 0.10 -Muestra la lista de directorios para la gestión de repositorio 0.12 – Indexa todos los paquetes de los "Package" en el repositorio origen y dado el campo "filename" importa los paquetes.	0.1-Elige la opción de crear nuevo repositorio en la interfaz. 0.3-Elige la opción "Crear repositorio a partir de otro existente". 0.5-Elige la opción "repositorio local" y luego "Copia" 0.7-El usuario elige en que directorio ubicar el origen nuevo repositorio. 0.9- Escoge el repositorio a copiar. 0.11 Selecciona el directorio destino.
		>>/home/user/Escritorio >>/etc/FTP >>/mnt/Datos	>>nova2011 >>nova >>cualquier_codename_existente	>>/home/user/Escritorio >>/etc/FTP >>/mnt/Datos							
EC_1.3 Crear repositorio de paquetes de software a partir de una copia a un repositorio remoto registrado en la base de datos	El usuario puede crear un repositorio local a partir de un repositorio remoto registrado en la base de datos.	URL	Herg_destino	-	-	-	-	-	-	0.2 - El sistema muestra las opciones para la creación de un nuevo repositorio. 0.4- Muestra opciones de importación. 0.6 -El sistema consulta la base de datos buscando los directorios disponibles para la gestión de repositorio y los muestra. 0.8- Descarga el "Packqge", "Crea un nuevo repositorio vacío EC_1.1". Luego incluye los paquetes indizados en el "Package".	0.1-Elige la opción de crear nuevo repositorio en la interfaz. 0.3-Elige la opción "Crear repositorio a partir de otro existente". 0.5 -Elige la opción "repositorio externo" y luego "Copia" 0.7 -El usuario elige en que directorio destino para el nuevo repositorio.
		>>http://nova.f10.u	>>/home/user/Escritorio >>/etc/FTP >>/mnt/Datos								

7.2 MEZCLAR REPOSITORIO.

SC <CP2. “Mezclar repositorio de paquetes de software”>

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios Reprepro.

El usuario debe estar bien identificado en el sistema.

Debe cargarse los datos de la base de datos pertenecientes a el directorio especificado y deben e interpretarse correctamente los ficheros de configuración de Reprepro ubicados en el directorio “/config” de los directorios Hergire existentes.

Debe mostrarse en la interfaz del visual una lista de repositorios existentes en el directorio especificado.

Debe estar instalado en el sistema operativo el servidor Apache, si se desea publicar el repositorio.

Debe existir una conexión de red en caso de trabajar con repositorios remotos.

Deben existir repositorios creados previamente.

RF18.Mezclar repositorio de paquetes de software.

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Respuesta del sistema	Flujo central
EC_2.1Mezclar repositorio de paquetes de software	*El usuario puede mezclar el repositorios de diferentes orígenes o tipos.*	Repositorio_ Origen_1 >>Repo(instancia)	Repositorio_ Origen_2 >>Repo (instancia)	Hergire_Destino >>Hergire_Class (instancia)	0.2- Muestra una lista de directorios en los cuales de puede gestionar repositorios. 0.4 - Muestra una lista de repositorios disponibles a ser mezclados. 0.6 - Muestra nuevamente la lista de repositorios disponibles a ser mezclados. 0.8 - Crea un nuevo repositorio en el directorio especificado e incluye los paquetes de los dos repositorios mezclados.	0.1 - Elige la opción 'gestionar repositorio' y luego 'Mezclar repositorio'. 0.3 - Elige de la lista el directorio destino de los repositorios a mezclar. 0.5 - Selecciona el primer repositorio origen para ser mezclado. 0.7 - Selecciona el segundo repositorio origen para ser mezclado.

Tabla 17: Diseño del caso de prueba para el requisito “Mezclar repositorios”.

7.3 ELIMINAR REPOSITORIO.

SC <CP3. "Eliminar repositorio">

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios Reprepro

El usuario debe estar bien identificado en el sistema.

Debe haberse cargado los datos de la base de datos pertenecientes a el directorio especificado y deben haberse interpretado correctamente los ficheros de configuración de Reprepro ubicados en en "/config" del directorio en cuestión.

RF2, RF5, RF8 Eliminar repositorio de paquetes de software

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central
EC 3.1 Eliminar repositorio de software	El actor puede eliminar cualquier repositorio. Accede al directorio donde se encuentre. Y eliminarlo. El repositorio no se eliminara físicamente del sistema, solamente no se gestionara más.	Hergire	Repositorio	0.2 Le muestra al usuario los diferentes directorios "Hergire" existentes en el disco del servidor. 0.4 Muestra una lista con los nombres clave, arquitecturas y componentes de los repositorios contenidos por el directorio seleccionado. 0.6 El sistema elimina el repositorio de los ficheros de configuración de reprepro. 0.7 Regresa al usuario a la vista de gestión de repositorios.	0.1 El usuario accede a las opciones de gestión para repositorios y selecciona eliminar repositorio local. 0.3 El usuario selecciona el directorio donde se encuentra el repositorio buscado. 0.5 Selecciona el repositorio y presiona "Enter".
		V	V		
		>>Hergire_Class (instancia)	>>Repositorio (instancia)		

Tabla 18: Diseño del caso de prueba para el requisito "Eliminar repositorio"

7.4 ACTUALIZAR REPOSITORIO.

SC <CP4. "Actualizar repositorio">

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios Reprepro

El usuario debe estar bien identificado en el sistema.

Debe cargarse los datos de la base de datos pertenecientes a el directorio especificado y deben e interpretarse correctamente los ficheros de configuración de Reprepro ubicados en el directorio "/config" de los directorios Hergire existentes.

RF3, RF6 y RF9 <Actualizar repositorio de paquetes de software>

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central
EC 4.1 Actualizar repositorio de paquetes de software.	En sistema copia las diferencias entre un repositorio origen y otro destino.	Repositorio_Orig	Repositorio_Dest	0.2 - Muestra una lista con todos los repositorios disponibles a actualizar. 0.4 – Muestra una lista de los repositorios remotos agregados a la base de datos. 0.6 – Comienza a descargar del repositorio origen solo los paquetes inexistentes y con una versión superior a los existentes en el repositorio destino. 0.7 – Regresa al usuario a la vista de gestión de repositorio.	0.1 – El usuario selecciona en el menú la opción "gestionar repositorio" y luego "Actualizar repositorio". 0.3 – Selecciona el repositorio local a actualizar. 0.5 – Selecciona el repositorio que va a ser origen de la actualización.
		V	V		
		>>Repositorio (instancia)	>>Repositorio_Remoto(instancia)		

Tabla 19: Diseño del caso de prueba para el requisito "Actualizar repositorio".

7.5 Crear mantenedor

SC <CP5. "Crear mantenedor de paquetes de software">

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios GnuPG
El administrador debe estar bien identificado en el sistema .

RF13 yRF15 "Crear mantenedor de paquetes de software" y "Crear llave de mantenedor"

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5	Respuesta del sistema	Flujo central
		Nombre	Apellidos	Email	ClaveGPG	ID_sistema		
EC_5.1 Crear mantenedor de paquetes de software	*Agregar un mantenedor de paquetes a la base de datos y generar su llave gpg*	>>Tomás >>rosco2011 >>cualquier_ combinación _alfanuméric a	>>Peña xxx >>cualquier_ combinación _alfanuméric a	>>nova2011@uci.cu >>tcabrales@estudiantes.uci.cu >>cualquier_combinación_alfanumérica+@+cadena alfanumérica+. +cadena	>>rootmaster >>xxx >>Cualquier_combinación_alfanumérica	>>nova >>tommy >>lolo >>cadena alfanumérica	0.2- Muestra una lista de los directorios mantenedores. Muestra también las opciones de gestión de repositorio. 0.4 - Pide al usuario introducir los datos del nuevo mantenedor. 0.6 - Crea la nueva llave GPG en el sistema, obtiene el nombre de la llave una vez generada y guarda los datos en la base de datos. (La generación de la llave GnuPG suele demorar algún tiempo) 0.7 – Retorna al usuario a la pantalla principal	0.1 - Elige la opción 'gestionar mantenedores' . 0.3 - Elige el la opción 'Crear nuevo mantenedor'. 0.5-introduce los datos requeridos

Tabla 20: Diseño del caso de prueba para el requisito " Crear mantenedor de paquetes de software".

7.6 Eliminar mantenedor

SC <CP6. "Eliminar mantenedor de paquetes de software">

Condiciones de ejecución

Debe estar instalado en el sistema operativo el software para la manipulación de repositorios GnuPG
El usuario debe estar bien identificado en el sistema y haber señalizado debidamente el directorio donde va a crear el repositorio de paquetes fuentes de software.
Debe mostrarse en la interfaz del visual una lista de repositorios existentes en el directorio especificado.
Debe estar instalado en el sistema operativo el servidor Apache, si se desea publicar el repositorio.

RF14, RF16. "Eliminar mantenedor de paquetes" y "Eliminar llave de mantenedor"

Escenario	Descripción	Variable 1	Variable 2	Variable 3	Variable 4	Variable 5	Respuesta del sistema	Flujo central
		Nombre	Apellidos	Email	ClaveGPG	ID_sistema		
EC_6.1 eliminar mantenedor de paquetes de software	Elimina un mantenedor de paquetes de la base de datos y su llave de GnuPG	>>Tomás >>rosco2011 >>cualquier_ combinación _alfanuméric a	>>Peña xxx >>cualquier_ combinación _alfanuméric a	>>nova2011@uci.cu >>tcabrales@estudiantes.uci.cu >>cualquier_combinación_alfanumérica+@+cadena alfanumérica	>>rootmaster >>xxx >>Cualquier_combinación_alfanumérica	>>nova >>tommy >>lolo >>cadena alfanumérica	0.2 Muestra las opciones de gestión de mantenedores. 0.4Muestra una lista de los mantenedores registrados en la base de datos. 0.6- Elimina al mantenedor de la base de datos.	0.1 - Elige la opción 'gestionar mantenedores' . 0.3 - Elige el la opción 'Eliminar mantenedor'. 0.5 - Elige el mantenedor a eliminar.

Tabla 21: Diseño del caso de prueba para el requisito " Eliminar mantenedor de paquetes de software".

7.7 CHEQUEAR INTEGRIDAD DE REPOSITORIOS

SC <CP7. Chequear integridad de repositorios>

Condiciones de ejecución

El administrador debe estar bien identificado en el sistema.

Debe cargarse al menos uno de los directorios "Hergire" contenidos en la base de datos.

Debe existir al menos un repositorio el cual realizar chequeos.

Debe estar especificado en las configuraciones, los datos correctos correspondientes al servidor SMTP para realizar las notificaciones.

RF14, RF15 y RF16 "Brindar informe de repositorio", "Alertar mal funcionamiento de repositorio"y "Enviar correo a mantenedor".

Escenario	Descripción	Variable 1	Variable 2	Respuesta del sistema	Flujo central
EC 1.1 Se encuentran errores	El administrador a través de este requisito. Puede escanear todos los repositorios del sistema en busca de errores. Una ves encontrados los errores son mostrados y se brinda la opción de ser notificados los mantenedores responsables de los problemas existentes en los repositorios.	-	-	0.2 – Revisa todas las listas de índices disponibles en los repositorios registrados en busca de referencias rotas. 0.3 – Muestra los errores encontrados y pregunta si desea que sean notificados. 0.5 – Busca el email de cada uno de los mantenedores responsables y envía un correo electrónico con los datos del error. 0.6 – Muestra la pantalla principal.	0.1 – El usuario elige la opción "Chequear integridad". 0.4 – Introduce "si" en caso que se desee notificar a los mantenedores responsables.
		-	-		
EC 1.2 No se encuentran errores.	En caso de que no se detecten errores	-	-	0.2 – Revisa todas las listas de índices disponibles en los repositorios registrados en busca de referencias rotas. 0.3 – Aparece notificación de la inexistencia de errores. 0.5 – Retorna al usuario a la vista principal.	0.1 – El usuario elige la opción "Chequear integridad". 0.4 – Presiona la tecla "Enter" para continuar.
		-	-		

Tabla 22: Diseño del caso de prueba para el requisito RF14, RF15 y RF 16.

Anexo 8: Diagramas de colaboración

8.1 Mezclar repositorio

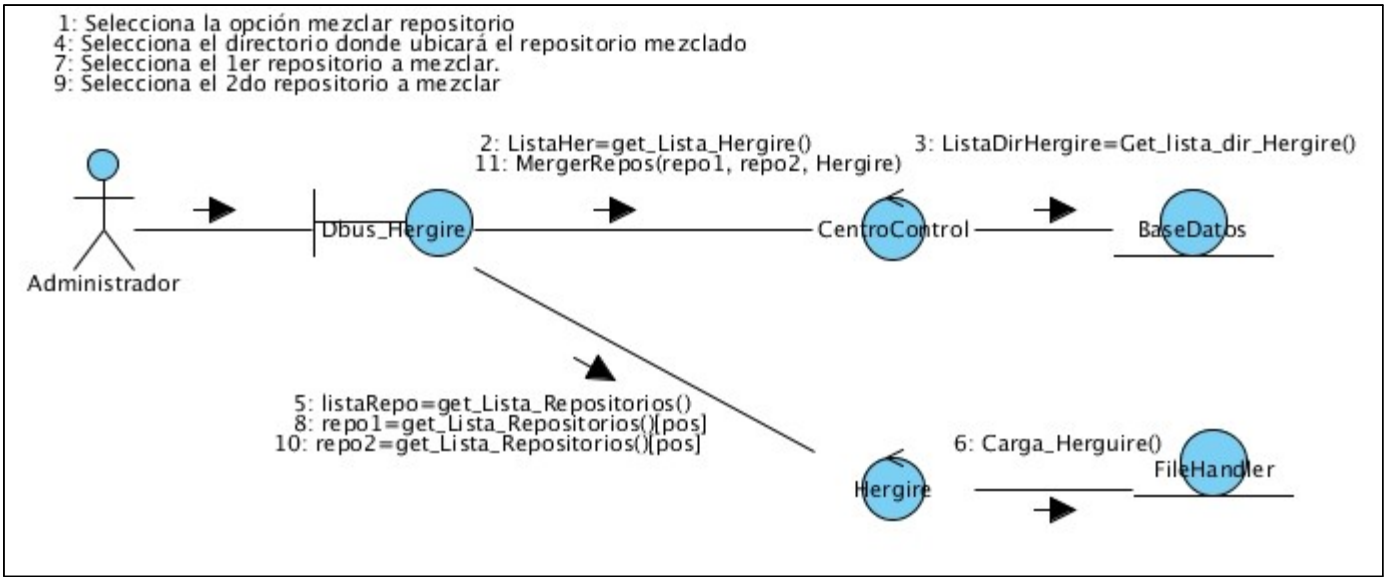

Figura 18: Muestra la implementación de el requisito funcional “Mezclar repositorio”.

8.2 Crear repositorio

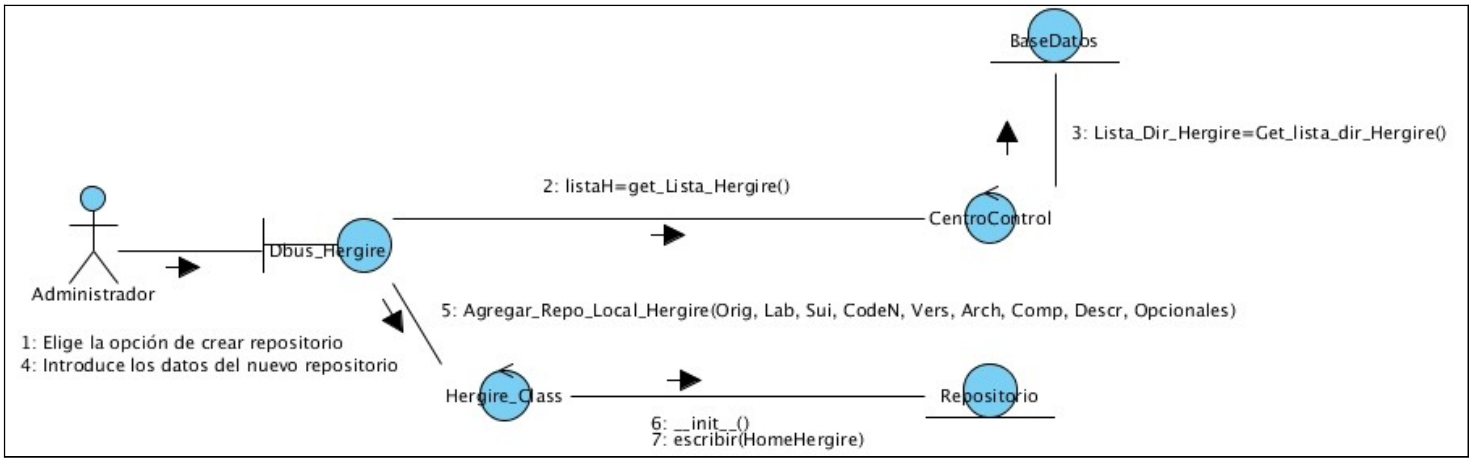

Figura 19: Muestra la implementación de el requisito funcional “Crear repositorio”.

8.3 Eliminar mantenedor

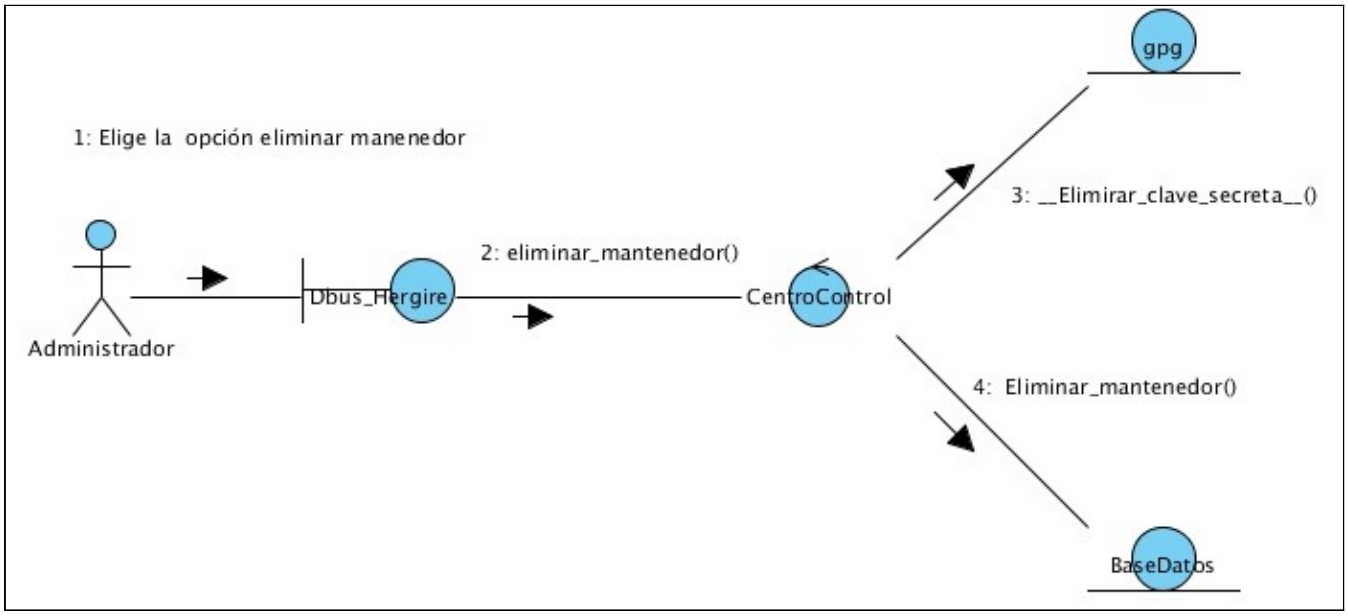

Figura 20: Muestra la implementación de el requisito funcional "Eliminar mantenedor".

8.3 Crear mantenedor

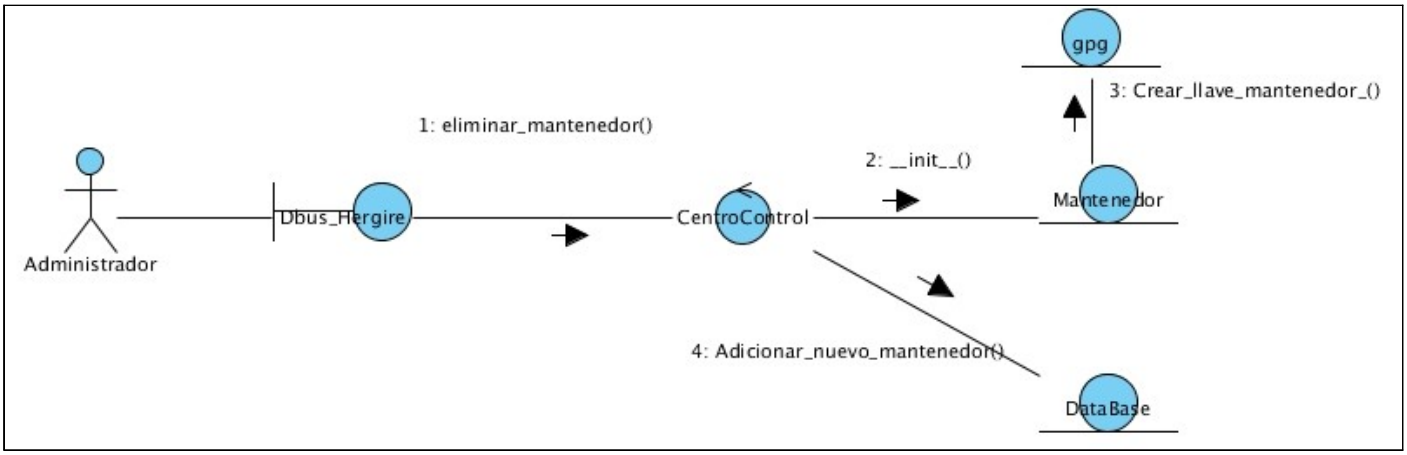

Figura 21: Muestra la implementación de el requisito funcional "Crear mantenedor".