Universidad de las Ciencias Informáticas

Facultad 1

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

Título:

Herramienta para gestionar los repositorios locales de *software* en Nova.

Autor:

Alejandro Tapia Chiroldes

Tutores:

MSc. Dariem Pérez Herrera

Ing. Héctor Pérez Baranda

La Habana, Cuba.

Junio de 2014.

Pensamiento

See the glass as half full. Understand that as you try to fill it, the glass will get bigger.

Taken from "Zen Guitar" by Philip Toshio Sudo

Declaración de autoría

Declaro ser el único autor de este trabajo y concedo 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 2014.

_____

Alejandro Tapia Chiroldes

_____ _____

MSc. Dariem Pérez Herrera Ing. Héctor Pérez Baranda

Dedicatoria

A mi familia por soportarme y apoyarme, en este aspecto especialmente a mi Titirriti que fue la que más lo sufrió.

A todos esos amigos que nacieron de una guitarra y una taza de café en largas noches de desvelo, gracias a ellos aprendí a ver en colores.

A todas esas personas que de una manera u otra me dieron un empujón para salir adelante cuando todavía no sabía ni cómo empezar a escribir este documento, y mucho menos, definir el nombre la clase principal de mi aplicación.

Agradecimientos

A mi familia por todo el apoyo, la confianza y la paciencia.

En especial a todos esos que corrigieron código y letras de esta investigación.

Resumen

En la Universidad de las Ciencias Informáticas (UCI), en el Centro de Soluciones Libres (CESOL), el departamento de Sistema Operativo se encarga del desarrollo de la distribución GNU/Linux Nova. Esta permitirá, con la entrega de un sistema operativo estable y factible de utilizar, sustentar todas las funcionalidades necesarias para lograr la soberanía tecnológica en nuestro país. Las distribuciones basadas en GNU/Linux trabajan bajo el principio de la dependencia de un repositorio de aplicaciones, en el cual se encuentran todos los programas y actualizaciones disponibles. Para garantizar un despliegue exitoso y altos índices de aceptación por los futuros usuarios, se hace necesario un sistema capaz de gestionar y configurar los repositorios de *soft*ware en Nova de manera local, prescindiendo de la conectividad a una red con acceso a los mismos. El objetivo central de este trabajo, es el análisis, diseño e implementación de un sistema informático, que permita realizar el proceso de gestión y configuración de repositorios locales en Nova.

Palabras clave: conectividad, repositorios, gestión, local.

Índice

Introducción....1
Capítulo 1. Fundamentación teórica....5
1.1 Introducción....5
1.2 Conceptos Fundamentales....5
1.2.1 Repositorio....5
1.2.2 Paquetes....5
1.2.2.1 Paquete fuente....5
1.2.2.2 Paquete binario....6
1.3 Análisis de sistemas para la creación y gestión de repositorios personalizados....6
1.3.1 Portable Software Center (PSC)....7
1.3.2 APTonCD....7
1.3.3 Keryx....8
1.3.4 YAST....9
1.3.5 Centro de Software de Ubuntu....10
1.3.6 Sistema para la creación de repositorios personalizados en línea (REPPER)....10
1.4 Valoración del estudio realizado....11
1.5 Metodología de desarrollo de software....12
1.6 Lenguajes, tecnologías y herramientas de desarrollo utilizadas....12
1.6.1 Lenguajes....13
1.6.2 Plataforma de desarrollo....15
1.6.3 Herramientas CASE....15
Capítulo 2. Análisis y diseño de la solución propuesta....17
2.1 Introducción....17
2.2 Propuesta del sistema a desarrollar....17

2.3 Modelo de Dominio....18
2.4 Diagrama de Clases....20
2.5 Casos de uso....22
2.6 Requisitos no funcionales....34
2.7 Diagrama de Secuencia (DS)....34
2.8 Descripción de la arquitectura....37
2.8.1 Estilo arquitectónico....37
2.8.2 Patrones de diseño....38
2.9 Conclusiones parciales....39
Capítulo 3. Implementación y prueba al sistema....40
3.1 Introducción....40
3.2 Diagrama de Componentes....40
3.3 Pruebas de Software....41
3.3.1 Pruebas de Aceptación....41
3.4 Descripción de los principales métodos implementados....46
3.5 Conclusiones Parciales....48
Conclusiones....49
Recomendaciones....50
Referencias bibliográficas....51
Bibliografía consultada....53
Anexos....56
Glosario de términos....62

Índice de Tablas

Tabla 1: Descripción de objetos o relaciones que intervienen en el proceso de implementación..19
Tabla 2: Especificación de requisitos....24

Tabla 3: Crear repositorio.....25
Tabla 4: Eliminar repositorio.....26
Tabla 5: Adicionar paquete.....27
Tabla 6: Eliminar paquete.....28
Tabla 7: Configurar repositorio.....29
Tabla 8: Detectar dispositivos extraíbles.....30
Tabla 9: Importar repositorio.....31
Tabla 10: Exportar repositorio.....32
Tabla 11: Instalar paquete.....33
Tabla 12: CPA_01: Gestionar repositorio.....42
Tabla 13: CPA_02: Gestionar repositorio.....43
Tabla 14: CPA_01: Gestionar paquete.....44
Tabla 15: CPA_02: Gestionar paquete.....45

Índice de Ilustraciones

Ilustración 1: Modelo de Dominio.....19
Ilustración 2: Diagrama de Clases.....21
Ilustración 3: DS_Crear repositorio.....35
Ilustración 4: DS_Eliminar repositorio.....35
Ilustración 5: DS_Agregar paquete.....36
Ilustración 6: DS_Eliminar paquete.....36

Índice de Anexos

Anexo 1: CPA_01: Configurar repositorio.....56
Anexo 2: CPA_01: Detectar dispositivo extraíble.....56
Anexo 3: CPA_02: Importar repositorio.....57
Anexo 4: CPA_02: Exportar repositorio.....58
Anexo 5: CPA_01: Instalar paquete.....58
Anexo 6: DS: Configurar repositorio. Detectar repositorio.....59
Anexo 7: DS: Importar repositorio.....59
Anexo 8: DS: Exportar repositorio.....60

Índice

Anexo 9: DS: Instalar paquete.....60
Anexo 10: Diagrama de Casos de Uso.....61
Anexo 11: Arquitectura N-Capas (3 Capas).....61

Introducción

En la Universidad de las Ciencias Informáticas (UCI), en el Centro de Soluciones Libres (CESOL), el departamento de Sistema Operativo se encarga del desarrollo de la distribución cubana de GNU/Linux Nova. La entrega de esta distribución permitirá sustentar todas las funcionalidades necesarias para lograr la soberanía tecnológica en nuestro país. Permitirá, con su despliegue, sustituir el uso de gran parte de las aplicaciones privativas que son usadas en Cuba, tanto en el sector estatal como particular.

Las distribuciones basadas en GNU/Linux trabajan bajo el principio de la dependencia de un repositorio de aplicaciones, en el cual se encuentran todos los programas y actualizaciones disponibles, esto permite que cada usuario, de manera eficiente, pueda acceder a esta información y hacer uso de la misma. Estos repositorios para garantizar el acceso de todos los usuarios, se encuentran alojados en un servidor en la red, permitiendo así que se hagan múltiples peticiones al mismo simultáneamente, por lo que se hace imprescindible la existencia de una infraestructura de red potente para satisfacer esa demanda. Esta característica puede ser considerada como una debilidad para el despliegue de esta distribución, debido a que nuestro país carece de la tecnología adecuada para soportar este fenómeno a gran escala.

Una alternativa para prescindir de la conexión a una red con acceso a los repositorios sería descargarlos en su totalidad hacia un dispositivo de almacenamiento masivo y trasladarlos hacia las computadoras sin conectividad. Un repositorio de cualquiera de las distribuciones de GNU/Linux puede ocupar una media de 80 GigaBytes de espacio en disco duro, con posibilidades de aumentar debido a la aparición de nuevas aplicaciones. Además, considerando que no todas las aplicaciones ni dependencias de paquetes son usadas en su totalidad por todos los usuarios, sería una gran pérdida de recursos para el país llevar a cabo esta estrategia, debido en parte a la cantidad de dispositivos de almacenamiento que sería necesario adquirir y por otro lado, a que no todos los usuarios utilizarían estos dispositivos en su totalidad.

Otra característica de estos sistemas lo constituye el proceso de configuración de los repositorios en las computadoras, tarea que se realiza de manera manual mediante la modificación de un fichero del sistema o mediante el uso de herramientas tales como, *apt*, para los sistemas

basados en *Debian* y *yum*, para los basados en *Red Hat,* que aún siguen siendo complicados de utilizar por usuarios inexpertos.

A partir de lo anteriormente planteado, se identifica el siguiente **problema científico:** ¿Cómo garantizar la gestión de los repositorios de *software* en Nova de manera local en ausencia de una conexión de red con acceso a los mismos?

Una vez identificado el problema se establece como **objetivo general**: desarrollar una herramienta que garantice la gestión de repositorios de *software* de manera local para la distribución de GNU/Linux Nova.

El **objeto de estudio** de la presente investigación lo constituye el proceso de gestión de repositorios.

El **campo de acción** se identifica en la gestión de repositorios de manera local.

En el marco de esta investigación se ha desglosado el objetivo general en los siguientes **objetivos específicos**:

- Sistematizar el estudio sobre las distintas herramientas que existen para la gestión de repositorios locales y sobre su usabilidad en sistemas operativos basados en GNU/Linux.
- Analizar y diseñar una herramienta de gestión de repositorios locales de *software* en Nova.
- Implementar y validar una herramienta de gestión de repositorios locales de *software* en Nova.

Idea a defender:

Con el desarrollo de una herramienta que permita gestionar el repositorio de Nova de manera local podría facilitarse su configuración en todas las estaciones de trabajo.

Para dar cumplimiento a los objetivos específicos se planifican las siguientes **tareas de investigación**:

- Revisión de la bibliografía relacionada con los sistemas de gestión de repositorios locales para una mejor comprensión del proceso.
- Descripción de los requisitos tanto funcionales como no funcionales para facilitar la implementación de la solución propuesta.

- Diseño de la arquitectura de la herramienta a implementar para cumplir con los requisitos establecidos.
- Diseño de las pruebas a la herramienta para verificar su correcto funcionamiento.

En el desarrollo de la investigación se utilizan los siguientes **métodos científicos**:

Métodos teóricos:

- **Analítico-Sintético:** Posibilitó el análisis de fuentes relevantes relacionadas con los sistemas de gestión de repositorios, a partir de su estudio permitió definir las características fundamentales para la ejecución de la herramienta.
- **Inductivo-Deductivo:** Permitió extraer los métodos y funcionalidades de diferentes generadores de repositorios así como sus patrones de diseño para resolver problemas particulares de la solución en desarrollo.

El documento se encuentra estructurado en 3 capítulos, la fundamentación teórica, el análisis y diseño de la solución propuesta y la implementación y pruebas al sistema. Las conclusiones generales, la bibliografía general utilizada, el glosario de términos y los anexos sirven de sustento para lograr la total comprensión de la investigación. La estructura de los capítulos se define a continuación:

Capítulo 1: Fundamentación teórica

En este capítulo se hace referencia a todos los elementos teóricos a tener en cuenta en la investigación del sistema a implementar, haciendo énfasis en los principales sistemas de gestión de repositorios locales existentes. Además se analizan mecanismos, técnicas y lenguajes para dar solución a la problemática propuesta.

Capítulo 2: Análisis y diseño de la solución propuesta.

Se hace alusión a la solución propuesta, planteándose cómo se va a desarrollar la misma y las herramientas que se utilizarán para su implementación. Además, se describe el proceso ágil basado en la metodología OpenUp.

Capítulo 3: Implementación y pruebas.

Este capítulo está enfocado en la implementación y pruebas de las funcionalidades del sistema. Se completa el desarrollo del sistema basado en la arquitectura definida.

Capítulo 1. Fundamentación teórica

1.1 Introducción

En este capítulo son considerados algunos aspectos teóricos acerca de los repositorios y los paquetes en las distribuciones GNU/Linux, que servirán para dar mayor claridad en el cumplimiento del objetivo general planteado. Se exponen conceptos para lograr una mejor comprensión sobre los términos tratados en la investigación, así como un estudio de homólogos sobre los sistemas a cargo de estas funcionalidades en otras distribuciones basadas en *software* libre. Por último, se presentan, tanto la metodología como las herramientas y tecnologías a utilizar en la implementación de la solución propuesta.

1.2 Conceptos Fundamentales

A continuación se definen algunos conceptos asociados a la investigación que permitirán una mejor comprensión del documento.

1.2.1 Repositorio

En Linux, un repositorio no es más que un conjunto o recopilación de programas preparados para su instalación en una distribución determinada. Suelen ser archivos binarios precompilados, aunque también existe la posibilidad de descargarse el código fuente (1).

1.2.2 Paquetes

Un paquete es esencialmente una colección de archivos construidos en un único fichero, el cual puede ser manejado más fácilmente. Así como los archivos requeridos por el programa para ejecutarse, habrá también unos archivos especiales llamados “*scripts* de instalación”, los cuales copian los archivos en el lugar adecuado (2).

1.2.2.1 Paquete fuente

Los paquetes fuente son sencillamente paquetes que incluyen código fuente, y generalmente pueden ser utilizados por cualquier tipo de máquina si el código se compila de manera correcta (2).

1.2.2.2 Paquete binario

Los paquetes binarios son los que están construidos específicamente para algún tipo de ordenador o “arquitectura”. Ubuntu soporta las arquitecturas x86 (i386 o i686), AMD64 y PPC. Los paquetes binarios correctos se utilizarán automáticamente (2). Estos paquetes contienen código de máquina, no código fuente, por eso cada tipo de procesador necesita su propia versión de cada uno. Al existir varias distribuciones de GNU/Linux existen varios tipos de paquetes binarios.

Los paquetes de formato *deb*[1] y *rpm*[2] son ejemplos de paquetes binarios que contienen los ficheros y librerías asociadas a una aplicación. Los de formato *deb* son utilizados por la distribución *Debian* y sus derivadas. Es un contenedor de aplicaciones que facilita en gran medida la actualización del sistema. El mismo contiene la lista de las dependencias que este necesita para su funcionamiento. Los de formato *rpm* son utilizados por las distribuciones derivadas de *Red Hat*.

1.2.3 Dependencia de paquetes

Los programas a menudo utilizan archivos que son comunes para otras aplicaciones. En vez de poner esos archivos en cada paquete, se puede instalar un paquete separado para proporcionar esos archivos a todos los programas que los necesiten. Por eso, al instalar programas que necesitan esos archivos, el paquete que los contiene debe ser instalado. Cuando un paquete depende de otro de esa manera, esto se conoce como dependencia de paquete (2).

1.3 Análisis de sistemas para la creación y gestión de repositorios personalizados

A continuación se realiza el estudio de las herramientas encargadas de la creación de repositorios en la rama de paquetería de formato *deb*. La razón fundamental para este enfoque es que la solución es requerida por la distribución GNU/Linux Nova, la cual usa este tipo de paquetería, aunque no se obvian otras aplicaciones que puedan aportar requisitos tanto funcionales como no funcionales. Se proponen una serie de características y desventajas de las

1 *Deb*: Formato de paquetes de las distribuciones basadas en *Debian*.

2 *Rpm*: Formato de paquetes de las distribuciones basadas en Red Hat.

mismas, permitiendo arribar a conclusiones que consoliden el proceso de desarrollo de la herramienta.

1.3.1 Portable *Software* Center[3] (PSC)

Es un programa diseñado para crear repositorios personalizados de aplicaciones. El mismo es desarrollado por el proyecto Konoha, perteneciente a la comunidad de *software* libre de la UCI. Entre sus principales características podrían citarse las siguientes:

- Esta desarrollado sobre el lenguaje Python.
- El usuario puede ver información acerca de un paquete.
- Utiliza los repositorios que se encuentran configurados en el sistema.
- Implementa filtros para que las búsquedas de aplicaciones sean más rápidas.
- Contiene un campo de búsqueda con autocompletamiento.
- Está licenciado bajo GPLv3.

Sus desventajas para esta investigación son:

- Tiene problemas en la gestión de las dependencias de los paquetes a descargar.
- No permite gestionar varios repositorios portables simultáneamente.
- No crea los repositorios portables con la estructura clásica[4].
- El proyecto está detenido actualmente, con fecha de la última versión (27/11/2011) y los desarrolladores están trabajando en otros proyectos (3).

1.3.2 APTonCD

APTonCD es una herramienta con una interfaz gráfica que permite crear uno o más CD o DVD con todos los paquetes que se hayan descargado a través de *apt-get* o *aptitude*, creando un repositorio portable que se puede utilizar en otras computadoras. Entre sus principales características podrían citarse las siguientes:

- Permite realizar salvas de sus paquetes descargados mediante los gestores de paquetes *apt-get, aptitude o synaptic*.
- Utiliza los repositorios que se encuentran configurados en el sistema.

3 Centro de Software Portable, también conocido como repoman.

4 Estructura que tiene un repositorio oficial.

- Está licenciado bajo GPLv2.
- Comprime la imagen en formato ISO y brinda la opción de guardarlo en un soporte de almacenamiento (CD o DVD).

Sus desventajas para esta investigación son:

- Presenta problemas a la hora de gestionar las dependencias de los paquetes a incluir en el repositorio.
- Es necesario poseer privilegios administrativos para tener funcionalidades avanzadas.
- El proyecto está descontinuado actualmente, con fecha de última versión (14/03/2007) (4).

1.3.3 Keryx

Keryx es un gestor de paquetes pensado para descargar instaladores y usarlos posteriormente en varios equipos que no tengan acceso a la red. Con Keryx obtienes una lista de paquetes para instalar y actualizar programas. Keryx te brindará información de cada paquete y te permite buscar uno en particular(5).

Proporciona una interfaz gráfica que ayuda a instalar y actualizar *software*. Entre sus principales características podrían citarse las siguientes:

- Gestiona las dependencias de los paquetes a descargar.
- Utiliza los repositorios que se encuentran configurados en el sistema.
- Está escrito en lenguaje Python.
- Es multiplataforma.
- Está licenciado bajo GPLv3.

Sus desventajas para esta investigación son:

- No crea el repositorio con la estructura clásica.
- Solo está disponible en inglés.
- No funciona en GNU/Linux Nova debido a incompatibilidades entre las versiones del lenguaje Python que usan.
- La última versión tiene fecha (14/09/11).

1.3.4 YAST

Yast ("*Yet another system tool*" por sus siglas en inglés), no es sólo un *software* de instalación, sino también una herramienta que facilita la administración del sistema una vez desplegado, así como la instalación de *software* en el mismo (6). Es desarrollada principalmente en Perl y C++, con *frontends*[5] en Qt y *Curses*[6]. Está diseñado para trabajar sobre paquetería *rpm*, lo que lo hace incompatible con la paquetería utilizada por Nova, *deb*.

Entre sus características más representativas tenemos:

Instalación de *software* adicional: Cuenta con la posibilidad de almacenar *software* alternativo además del propio, esto le brinda al usuario una amplia gama de aplicaciones para instalar en el sistema. La aplicación está diseñada de tal manera que ordena los paquetes por categorías: Funciones de escritorio, Tecnologías base, Funciones de servidor, Entornos gráficos y otras. Esta alternativa permite desplegar sistemas personalizados desde el mismo momento de la instalación.

Prioridad de los repositorios: La instalación de los paquetes se puede gestionar por medio de un sistema de prioridades que se pueden asignar a los repositorios. La prioridad de un repositorio se determina por un valor entero entre 0 para la prioridad alta y 200 para la prioridad baja. Por defecto este valor es 99. Si un paquete se encuentra disponible en más de un repositorio, se usará aquel que tenga la prioridad más alta, en el caso que tengan la misma se usará el paquete más actualizado.

Habilitar los repositorios: Pueden ser añadidos tantos repositorios como se deseen pero solo se tendrán en cuenta los que estén marcados como "Habilitados". Podemos añadir un determinado repositorio para instalar un paquete en cuestión y una vez hecho eso deshabilitar el repositorio para que no lo tenga en cuenta pero conservándolo en la lista de repositorios disponibles en caso de que se desee usar en otro momento.

5 *Frontends*: Parte del *software* que interactúa con el usuario.

6 *Curses:* Librería añadida a disímiles lenguajes, que permite la representación de interfaces en terminales de consola como modo alternativo a la representación de solo texto.

1.3.5 Centro de *Software* de Ubuntu

El Centro de *Software* de Ubuntu es una aplicación de alto nivel para el manejo de las herramientas de administración de sistema **APT/DPKG**, está escrito en Python y es basado en **GTK+**[7]. Es un programa que nos permite buscar, instalar, y eliminar aplicaciones del sistema operativo, además permite añadir repositorios de terceros para instalar aplicaciones que no se encuentren en los repositorios oficiales de la distribución. Las aplicaciones se dividen en las siguientes categorías: Accesorios, Acceso Universal, Gráficos, Internet, Juegos, Oficina, Sonido y Video, Temas y ajustes, Ciencia e Ingeniería, Educación, Tipografías, Herramientas para desarrolladores y Sistema. Algunas de estas categorías cuentan con subcategorías para tener un mejor orden en su jerarquía de aplicaciones y facilitar su uso. Ofrece aplicaciones libres (o de código abierto), aplicaciones privativas y aplicaciones de pago. Surge con el objetivo de unificar los procesos de instalación, desinstalación, compra, manejo de repositorios, y actualización de aplicaciones de una manera simple y unificada en una sola aplicación general (7). El Centro de *Software* de Ubuntu, aunque no es una herramienta para la creación de repositorios locales, presenta características que pueden ser incluidas en la solución propuesta como, definir el proceso de listado de los paquetes por categorías y complementar la funcionalidad de instalación de aplicaciones.

1.3.6 Sistema para la creación de repositorios personalizados en línea (REPPER)

REPPER es un sistema para la creación de repositorios personalizados en línea, funciona bajo el concepto de "*software* como servicio"[8], fue desarrollado en el Centro de Soluciones Libres, perteneciente a la Universidad de las Ciencias Informáticas. Entre sus características pueden citarse las siguientes:

- El usuario puede ver información acerca de un paquete seleccionado.
- Permitir la realización de búsquedas de paquetes en el repositorio.
- Crear los repositorios con la estructura clásica.
- Utilización de repositorios que no se encuentren registrados.

7 GTK+:Conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario.

8 Modelo de distribución de *software* en donde una compañía provee el servicio de mantenimiento, operación diaria y soporte del *software* usado por el cliente.

Sus desventajas para esta investigación son:

- Requiere de una conexión permanente en cada computadora cliente para acceder al repositorio.
- Está desarrollada en lenguaje *Hypertext Pre-processor* (PHP) (3).

1.4 Valoración del estudio realizado.

Una vez concluido el estudio de los sistemas para la creación de repositorios locales, se puede constatar que aunque poseen similitudes en varias de sus funcionalidades, la gestión de dependencias de los paquetes no funciona correctamente en algunos de ellos, tal es el caso de APTon-CD y PSC. En el caso de Keryx además de no crear el repositorio de manera organizada y de estar disponible solo en inglés, no funciona sobre GNU/LINUX Nova, esto se debe a un problema de compatibilidad de las versiones de Python que utilizan. Aunque Keryx y PSC están escritos en Python y satisfacen gran parte de los requerimientos, fueron desechados debido a las características antes expuestas, además de estar obsoletos o inactivos, por lo que no se toman como punto de partida para el desarrollo de la solución propuesta. REPPER es una aplicación que satisface gran parte de las especificaciones propuestas en la investigación, pero al ser orientada hacia la web y requerir de una conexión de red permanente por las computadoras, entra en conflicto con los intereses de la investigación. En cuanto a YAST aunque se considera, entre las analizadas, la herramienta que mejor satisface nuestras necesidades, está diseñada para trabajar con paquetería *rpm*, haciendo esta incompatible con la paquetería que usa Nova, *deb*.

La solución propuesta no será meramente para la creación de repositorios locales, también permitirá su personalización, permitiendo a los usuarios seleccionar las aplicaciones deseadas para conformar su repositorio, pretende automatizar el proceso de detección de los mismos y cubrir todo el proceso de gestión y configuración.

Por lo antes expuesto se decidió desarrollar una nueva aplicación, en lugar de seleccionar una de las estudiadas. Se tendrán en cuenta las siguientes características para la creación del sistema propuesto:

- Creación de los repositorios locales manteniendo la estructura clásica.
- Gestionar las dependencias de los paquetes a descargar.

- Permitir la realización de búsquedas de paquetes en el repositorio.
- Instalación de paquetes.

1.5 Metodología de desarrollo de *software*

Una metodología de desarrollo de *software* no es más que un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. Es un acumulado de procedimientos, técnicas, artefactos y herramientas que guían a los desarrolladores en el proceso de realización de *software*.

Debido a que el equipo de desarrollo está compuesto por una sola persona, el tiempo de desarrollo es relativamente corto y los requisitos son cambiantes se optó por el uso de una metodología ágil. Se escogió como metodología a utilizar OpenUp, la cual es idónea para la entrega de productos en cortos períodos de tiempo; además de satisfacer las necesidades del proceso.

OpenUp es una metodología de desarrollo de *software*, de código abierto diseñada para pequeños equipos organizados, tomando una aproximación ágil del desarrollo. OpenUp está organizado dentro de cuatro áreas principales de contenido (8):

- **Concepción**: Definición de ¿qué se quiere hacer? y ¿para qué?, mediante la obtención de la suficiente información sobre lo que se debe hacer en el proyecto. Tiene como objetivo la captura de necesidades del cliente.
- **Elaboración**: Se determina la arquitectura y los posibles riesgos para la misma. Tiene como meta establecer la línea base para la elaboración de la arquitectura del sistema.
- **Construcción**: Enfocada al diseño, implementación y prueba de las funcionalidades del sistema. El propósito de esta fase es completar el desarrollo del sistema basado en la arquitectura definida.
- **Transición**: Asegurar que el sistema es entregado a los usuarios y estos evalúen sus funcionalidades y rendimiento.

1.6 Lenguajes, tecnologías y herramientas de desarrollo utilizadas

Con el objetivo de buscar la mejor alternativa en cuanto al lenguaje de programación, las tecnologías y las herramientas a utilizar, se realiza un estudio teniendo en cuenta las ventajas de cada una y que cumplan con las condiciones del sistema a desarrollar.

1.6.1 Lenguajes

Python: Lenguaje de programación utilizado en el desarrollo de la solución, es un lenguaje de alto nivel y que tiene entre sus premisas la organización y legibilidad del código, es un lenguaje multiparadigma, utiliza tipado dinámico, fuertemente tipado y multiplataforma. Es un lenguaje potente con una comunidad de desarrollo muy activa, que mantienen una cantidad de módulos y bibliotecas muy bien documentadas, lo que facilita el uso de estas en la implementación de las funcionalidades (9).

UML: Para el modelado de los elementos arquitectónicos fue utilizado el lenguaje UML que constituye el estándar para representación del desarrollo de *software*, ilustrando los procesos, objetos y entidades de un *software* así como sus relaciones.

D-bus: Es un mecanismo IPC[9] que alberga a un objeto que será exportado, se le asigna un nombre de bus mediante el cual será identificada y que el estándar plantea que en aras de que sea único se le pone un nombre muy parecido a un dominio DNS pero en sentido inverso "im.pidgin.purple.PurpleService" es el nombre de bus del cliente de mensajería pidgin. Dentro de este dominio se registran los objetos que serán invocados a través de un camino o path que adquiere una forma muy parecida a la del camino de un archivo en el sistema de archivos, "/im/pidgin/purple/PurpleObject".

Cada objeto exporta métodos y/o señales que pueden ser utilizados de forma remota. Esto se hace a través de interfaces, que tienen el objetivo de establecer un estándar de comunicación o un contrato de comportamiento de los objetos que las implementan. Las interfaces en D-Bus se identifican con nombres en un formato similar al de los nombres de bus de las aplicaciones, por ejemplo, una interfaz pudiera llamarse cu.uci.Ejecutable. Esta interfaz pudiera incluir varios métodos, por ejemplo, un método Run, el cual puede ser invocado de forma remota, o pudiera incluir señales, por ejemplo, una señal llamada EjecucionTerminada, la cual se puede poner a escuchar desde una aplicación cliente para ejecutar algo cuando esta sea emitida (10).

¿Porque usar D-bus?

9 IPC: Inter Process Comunication por sus siglas en inglés, Intercomunicación de procesos

D-bus utiliza un servidor centralizado que utiliza información de manera que la latencia en el flujo de la misma es muy baja, comparado con los demás mecanismos IPC. Brinda la libertad de no usar libdbus, una librería predeterminada para la comunicación entre aplicaciones, sino que brinda la posibilidad de poder escribir nuevas librerías, siempre y cuando estas cumplan las especificaciones requeridas, trabaja como un *framework* permitiéndole al sistema operativo extenderse con un servicio de mensajería aplicado a la comunicación entre un nuevo dispositivo y las aplicaciones del sistema. En cuanto la gestión de memoria, en caso de que se necesite compartir grandes cantidades, usando las primitivas de compartición se obtienen mejores resultados pero se debería programar o utilizar una librería portable de memoria compartida, D-bus realiza este proceso de manera nativa (10).

Python-apt: Proporciona acceso a casi todas las funciones soportadas por las librerías apt-pkg y apt-inst subyacentes. Esto significa que es posible volver a escribir los programas de *frontend* como apt-CD-ROM en Python. Al pasar por la biblioteca, los dos primeros módulos son apt_pkg y apt_inst. Estos módulos son enlaces directamente a las bibliotecas apt-pkg y apt-inst y la base para el resto de python-apt. El paquete apt utiliza apt_pkg y mod 'apt_inst' para proporcionar fáciles formas de manipular la memoria caché, obtención e instalación de nuevos *paquetes*. El último paquete es aptsources, este proporciona clases y funciones para leer archivos como '*/etc/apt/sources.list*' y modificarlos (11).

- apt_pkg - Los enlaces de bajo nivel para apt-pkg.
- apt_inst - Trabajando con los paquetes locales de *Debian*.
- apt.cache - La clase Cache.
- apt.cdrom - Funcionalidad como en apt-cdrom.
- apt.debfile - Clases relacionadas con los archivos del paquete *Debian*.
- apt.package - Las clases para el manejo de paquetería.
- apt.progress.base - Las clases abstractas para los informes de progreso.
- apt.progress.text - Información sobre los progresos de las interfaces de texto.

- apt.progress.gtk2 - Información sobre los progresos de las interfaces GTK+.
- aptsources.distinfo - Proporcionar información de metadatos para los repositorios de distribuciones.
- aptsources.distro - Abstracción y Distribución del sources.list.
- aptsources.sourceslist - Proporcionar una abstracción del sources.list.

Qt: biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con una interfaz gráfica de usuario, utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios lenguajes de programación a través de *bindings*[10]. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos. Distribuida bajo los términos de GNU *Lesser General Public License*, Qt es *software* libre y de código abierto (12).

1.6.2 Plataforma de desarrollo

Para el desarrollo utilizar el entorno de desarrollo Eclipse en su versión 3.7.2, que integrado a un conjunto de *Plug-Ins*, incluye una gran cantidad de funcionalidades utilizadas para el desarrollo de la solución. Se integró a esta plataforma el *Plug-In* PyDev que integra las funcionalidades de Eclipse para el desarrollo con Python.

1.6.3 Herramientas CASE

Visual Paradigm fue la herramienta CASE seleccionada, es una herramienta UML profesional que soporta el ciclo de vida completo de desarrollo del *software*: análisis y diseño, construcción, pruebas y despliegue. El lenguaje de modelado UML permite una más rápida construcción de aplicaciones además permite la generación automática de código. Permite realizar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (13).

1.7 Conclusiones parciales

En el presente capítulo fueron analizados diferentes sistemas encargados de la gestión de repositorios locales, donde se identificó que no cumplen con las necesidades actuales de la

10 Adaptación de una biblioteca para ser usada en un lenguaje de programación distinto de aquel en el que ha sido escrita.

distribución de GNU/Linux Nova. Por tanto se decide desarrollar un nuevo sistema para la creación y gestión de repositorios personalizados y locales, manteniendo algunas características de los sistemas estudiados. La metodología de desarrollo OpenUp permitió definir los artefactos necesarios para el desarrollo de la aplicación. Se seleccionó el lenguaje de programación *Python*. Para realizar la interfaz visual se decide utilizar Qt4 teniendo en cuenta que presenta un entorno gráfico más ligero que GTK+, permitiendo que aumente el rendimiento en estaciones de trabajo de pocas prestaciones.

Capítulo 2. Análisis y diseño de la solución propuesta

2.1 Introducción

Definidas la metodología, los lenguajes, las herramientas y tecnologías escogidas para el desarrollo de la solución propuesta se procede a la descripción de la misma. En el siguiente capítulo se definen y especifican los requisitos funcionales y no funcionales presentes en el sistema. Además la descripción de la arquitectura y el diseño. Para una mejor comprensión del ciclo de funcionamiento se generaron los diagramas de secuencia y de clases, así como el modelo de dominio del sistema.

2.2 Propuesta del sistema a desarrollar

La Herramienta para la Gestión de los Repositorios Locales y Portables (HGRP), es un sistema a partir del cual se podrán gestionar repositorios personalizados y locales para la distribución GNU/Linux Nova, sin necesidad de conectividad. Permitirá la gestión de varios repositorios locales provenientes de diferentes fuentes de actualización, cada repositorio podrá contener paquetes de diferentes arquitecturas y versiones, manteniendo la misma fuente externa; esto permitirá crear repositorios locales con paquetes de varias versiones de la distribución que se desee; se incluye además la instalación de paquetes y la configuración automática del repositorio en el sistema. Esto le brindará al usuario la posibilidad de gestionar tantos repositorios locales como desee, cada uno con las aplicaciones de acuerdo a sus necesidades, sin tener que utilizar un único repositorio en su totalidad.

La clase Principal se conecta a una fuente de actualización en busca de paquetes de *software* disponibles, dándole la posibilidad al usuario de seleccionar los que desee. La información de los paquetes es enviada a la clase LocalRepo, donde se gestionan las dependencias de cada uno para proceder a su descarga, y una vez concluido se le muestra al usuario el estado actual de su repositorio local con la lista de paquetes.

Se generará un archivo de registro de nombre *hgrp.json,* encargado de mantener la información de los repositorios locales existentes en el sistema. Contendrá la siguiente información de cada repositorio local:

- **Id:** Identificador del repositorio.

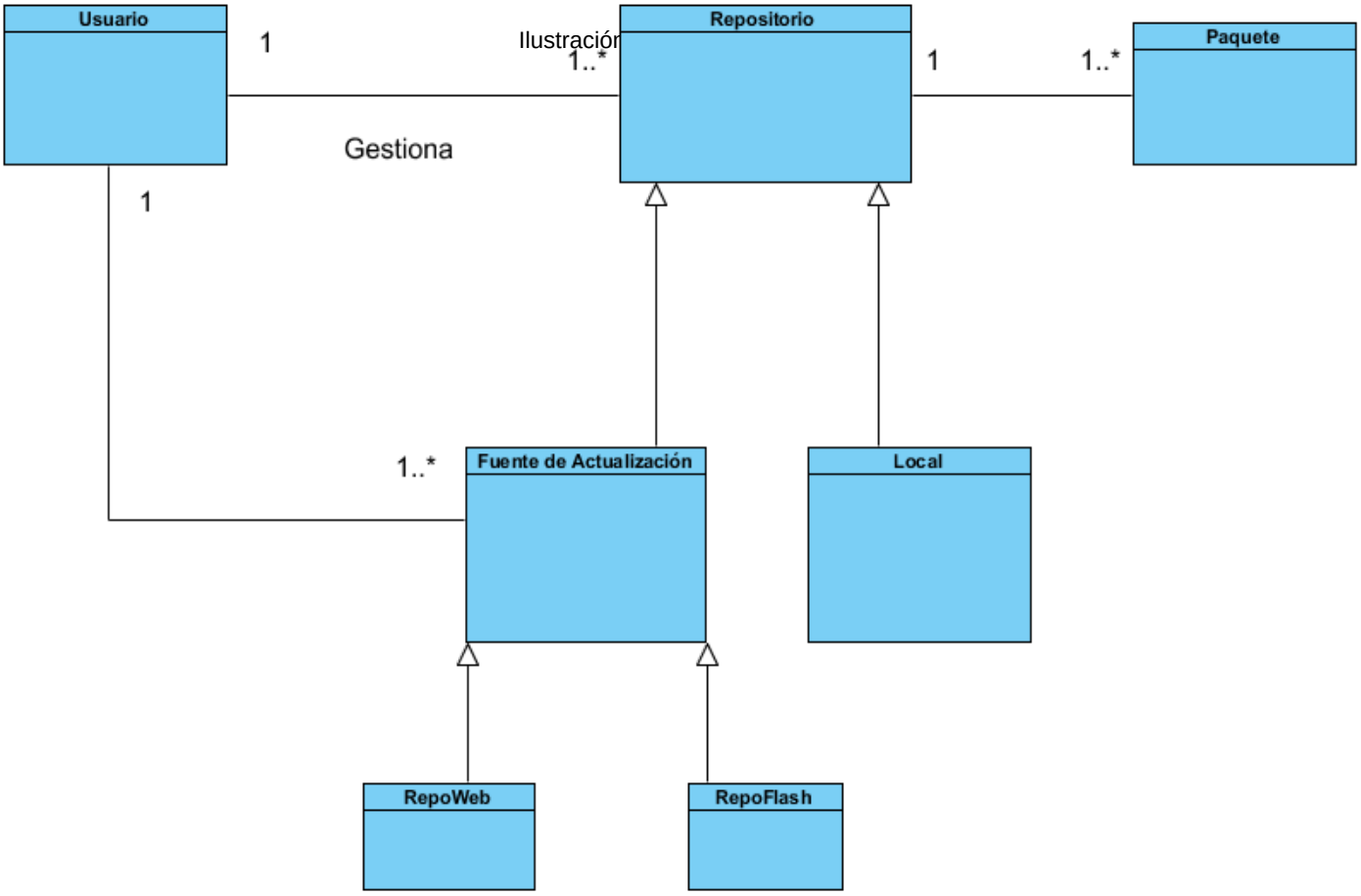
- **RepoLocal:** Nombre del repositorio local.
- **URL:** Fuente de actualización asociada a ese repositorio.
- ***CodeName:*** Nombre en código de la distribución.
- **Arquitectura:** Arquitectura de los paquetes presentes en el repositorio local.
- **ListaPaquetes:** Lista con el nombre de los paquetes contenidos en el repositorio.
- **Ruta:** Ruta donde está localizado dentro del sistema.

En cada repositorio local creado habrá una copia de esta información, a la hora de iniciar la aplicación, se tendrán en cuenta estos parámetros para confirmar la existencia de los repositorios locales creados en la sesión previa.

Con el objetivo de especificar el funcionamiento del sistema de manera general se muestra a continuación el modelo de dominio del sistema.

2.3 Modelo de Dominio.

El modelo de dominio es el encargado de representar los conceptos fundamentales para el desarrollo de una aplicación y las diferentes relaciones que existen entre ellos. Son presentados como clases los aspectos esenciales y las interrelaciones denotan una estructura de funcionamiento, brindan una mejor comprensión del medio actual para la concepción de un futuro sistema. La siguiente ilustración muestra el modelo de dominio del sistema a implementar.



Ilustración

A continuación se describen cada uno de los objetos o relaciones que intervienen en el proceso de implementación:

Objeto o relación	Descripción
Usuario	Persona que interactúa con la aplicación encargada de la gestión de repositorios.
Repositorio	Pueden ser locales o fuentes de actualización. Conformado por uno o muchos paquetes.
Fuente de actualización	Repositorio externo a partir del cual el usuario crea sus repositorios locales, puede encontrarse en la web o en un dispositivo extraíble.
Local	Repositorio creado por el usuario en su computadora.
RepoWeb	Repositorio localizado en la web.
RepoFlash	Repositorio localizado en un dispositivo extraíble.

Tabla 1: Descripción de objetos o relaciones que intervienen en el proceso de implementación.

2.4 Diagrama de Clases

A continuación se brinda una breve descripción de cada una de las clases en las que se dividió la herramienta:

Principal:

Clase encargada de interactuar con el usuario de manera directa mostrando el estado de los repositorios locales existentes y permitiendo realizar las acciones pertinentes, es la encargada además de hacer una salva de los repositorios existentes en el sistema para la posterior restauración del estado del mismo, así como permitir su modificación en tiempo de ejecución, mediante la clase Repositorio.

NuevoRepo:

Es la clase encargada de crear un nuevo repositorio local en el sistema mediante una instancia de la clase localRepo, la cual contiene los métodos necesarios para la creación y descarga de la caché, así como de los paquetes y sus dependencias.

DetectarDispositivo:

Demonio[11] que está a la espera de un nuevo dispositivo que se conecte vía *USB*, se encarga de su detección y validación. En caso de existir un repositorio, configura el sistema para su utilización y ejecuta la aplicación HGRP para posibilitar la gestión del mismo.

Las clases ***FlashScanner***, ***UrlScanner*** y ***Parser*** se encargan de la validación de las rutas introducidas por el usuario.

La clase **Importar** se encarga de importar un repositorio local ubicado en un dispositivo externo hacia la computadora.

La clase **Exportar** se encarga de exportar un repositorio local hacia una ubicación diferente, dentro o fuera del sistema.

11 Programa que se ejecuta en segundo plano, esperando a que ocurran determinados eventos.

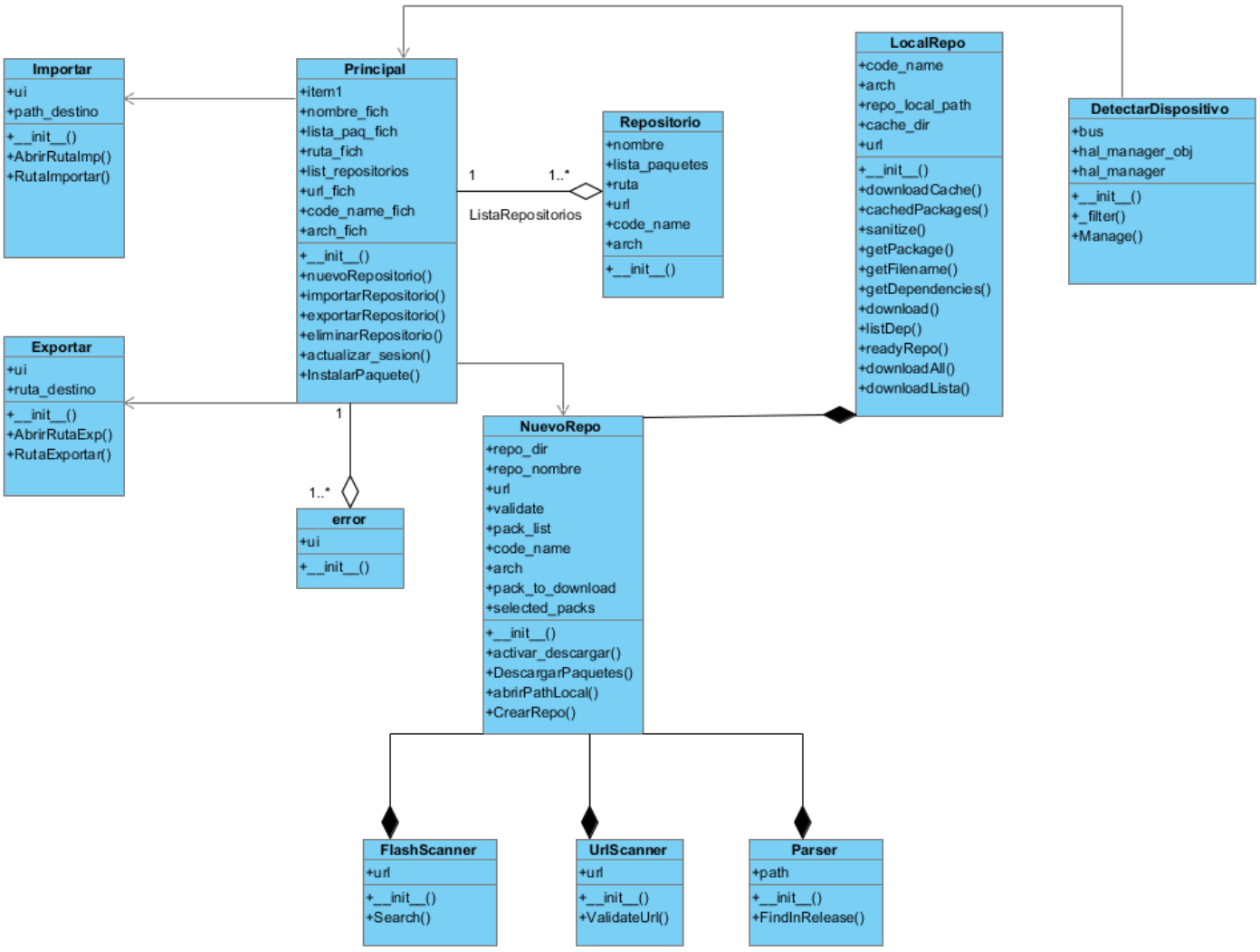


Ilustración 2: Diagrama de Clases

Partiendo del análisis del modelo de dominio y el diagrama de clases del sistema se identificaron las siguientes funcionalidades del sistema:

RF1: Crear repositorio

RF2: Eliminar repositorio

RF3: Agregar paquete

RF4: Eliminar paquete

RF5: Configurar repositorio

RF6: Detectar dispositivo extraíble

RF7: Importar repositorio

RF8: Exportar repositorio

RF9: Instalar paquete

Los RF quedaron agrupados en los siguientes casos de uso:

2.5 Casos de uso.

En este acápite se muestran los casos de uso identificados en el sistema con los requisitos funcionales que estos implementan. El diagrama de Casos de Uso del sistema puede verse en el Anexo 11.

CU1. Gestionar repositorio

RF1: Crear repositorio

RF2: Eliminar repositorio

CU2. Gestionar paquete

RF3: Agregar paquete

RF4: Eliminar paquete

CU3. Configurar repositorio

RF5: Configurar repositorio

CU4. Detectar dispositivo extraíble

RF6: Detectar dispositivo extraíble

CU5. Importar repositorio

RF7: Importar repositorio

CU6. Exportar repositorio

RF8: Exportar repositorio

CU7. Instalar paquete

RF9: Instalar paquete

La siguiente tabla muestra la especificación de los requisitos definidos anteriormente:

Nº	Nombre	Descripción	Complejidad	Prioridad para cliente
RF1	Crear repositorio.	Crear un repositorio local en la ruta especificada por el usuario manteniendo.	Alta	Alta
RF2	Eliminar repositorio	El sistema será capaz de eliminar el repositorio local escogido.	Alta	Alta
RF3	Adicionar paquete.	Adicionar paquetes en el repositorio local seleccionado.	Alta	Alta
RF4	Eliminar paquete.	Eliminar paquetes del repositorio local seleccionado.	Alta	Alta
RF5	Configurar el repositorio	La aplicación será capaz de configurar el repositorio de manera automática, agregando un fichero de nombre, *nombre_del_repositorio_sources.list* , con la configuración del repositorio local, en el directorio *'/etc/apt/sources.list.d/'* y hace un *update* del sistema.	Alta	Alta
RF6	Detectar dispositivo extraíble	Se iniciará la aplicación HGRP una vez detectado un dispositivo extraíble con un repositorio local creado por la aplicación.	Media	Media
RF7	Importar repositorio	Importar un repositorio de un dispositivo extraíble.	Media	Media
RF8	Exportar repositorio	Exportar el repositorio local a una ruta especificada por el usuario.	Media	Media
RF9	Instalar paquete	El sistema realiza una llamada al Centro de *Software* para permitir el proceso de instalación de paquetes.	Baja	Baja

Tabla 2: Especificación de requisitos

Descripción de requisitos:

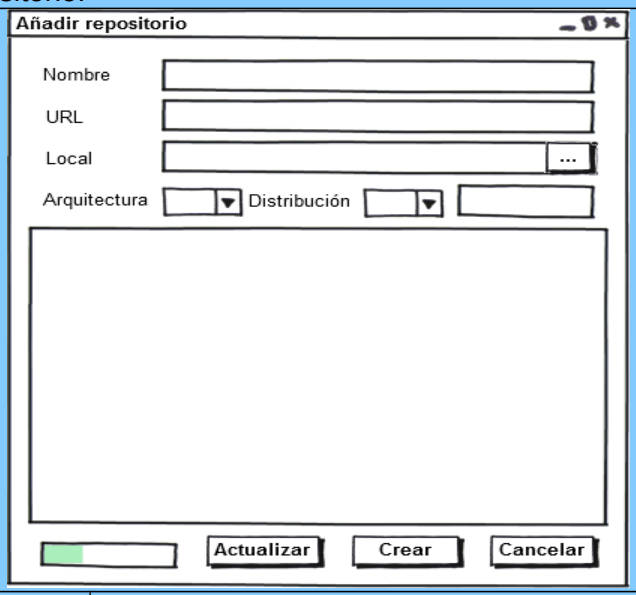
<table>
<tr><th colspan="3">RF1: Crear repositorio</th></tr>
<tr><td>Precondiciones</td><td colspan="2">Tener acceso a una fuente de actualización.</td></tr>
<tr><td>Flujo de eventos</td><td colspan="2"></td></tr>
<tr><td>Flujo básico</td><td colspan="2">Crear repositorio.</td></tr>
<tr><td>Prototipo</td><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Actor</td><td>Sistema</td></tr>
<tr><td>1.</td><td>1. El usuario inicia el caso de uso.</td><td>1. Muestra una ventana para insertar los datos del nuevo repositorio: nombre, fuente de actualización, ruta dentro del sistema, arquitectura y distribución.</td></tr>
<tr><td>2.</td><td>1. Inserta los datos.</td><td>1. Valida que en la fuente de actualización exista un repositorio.
2. Muestra una lista con los paquetes disponibles.</td></tr>
<tr><td>3.</td><td>1. Selecciona los paquetes.
2. Pulsa el botón Crear.</td><td>1. Crea el repositorio con los datos introducidos.
2. Actualiza el sistema.</td></tr>
<tr><td>4.</td><td></td><td>1. Fin del caso de uso.</td></tr>
<tr><td colspan="3">Post-Condiciones</td></tr>
<tr><td>1.</td><td colspan="2">Queda configurado el nuevo repositorio en el sistema.</td></tr>
<tr><td colspan="3">Validaciones</td></tr>
<tr><td>1.</td><td colspan="2">En el enlace http debe existir un repositorio.</td></tr>
<tr><td>2.</td><td colspan="2">El dispositivo extraíble debe contener un repositorio.</td></tr>
<tr><td>3.</td><td colspan="2">No puede contener tildes.</td></tr>
</table>

Tabla 3: Crear repositorio

RF2: Eliminar repositorio		
Precondiciones	Exista un repositorio local.	
Flujo de eventos		
Flujo básico	Eliminar repositorio.	
Actor		**Sistema**
1.	1. Inicia el caso de uso.	1. Muestra una lista con los repositorios locales.
2.	1. Selecciona uno o varios repositorios	1. Elimina los repositorios seleccionados. 2. Modifica el fichero *hgrp.json* y elimina los datos de los repositorios eliminados. 3. Actualiza el sistema.
4.		1. Fin del caso de uso.
Post-Condiciones		
1.	Se elimina el repositorio del sistema.	
Validaciones		
1.	Exista al menos un repositorio local en el sistema.	

Tabla 4: Eliminar repositorio

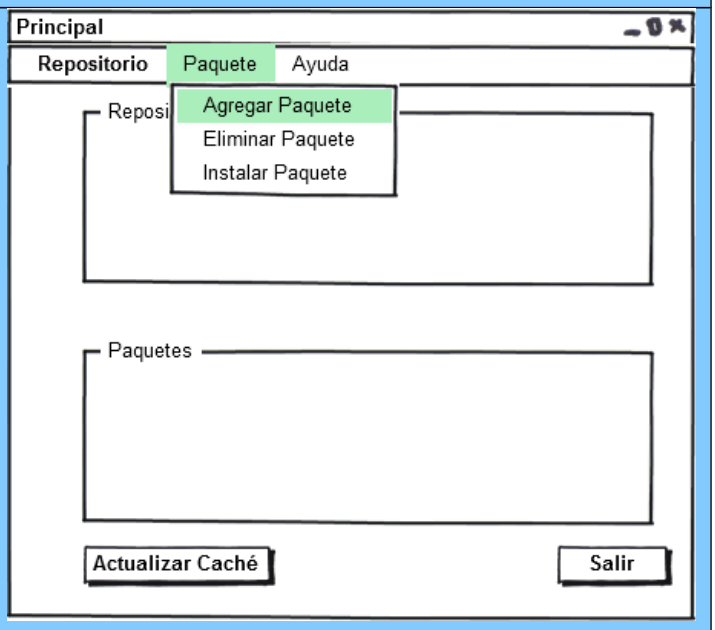
RF3: Adicionar paquete		
Precondiciones	Exista un repositorio local	
Flujo de eventos		
Flujo básico	Adicionar paquete	
Prototipo		
Actor		**Sistema**
1.	1. Inicia el caso de uso.	1. Muestra una lista con los repositorios locales disponibles.
2.	1. Selecciona el icono adicionar asociado al repositorio.	1. Muestra una ventana con los paquetes disponibles.
3.	1. Si lo deseas selecciona otra fuente de actualización.	1. Muestra una lista de paquetes disponibles en esa fuente de actualización.
4.	1. Selecciona los paquetes a adicionar.	1. Adiciona los paquetes seleccionados con todas sus dependencias al repositorio local. 2. Actualiza el sistema.
5.		1. Fin del caso de uso.
Post-Condiciones		
1.	Se agrega un nuevo paquete en el repositorio local.	
Validaciones		
1.	Exista una fuente de actualización disponible.	
2.	El repositorio a modificar tienes que ser local.	

Tabla 5: Adicionar paquete

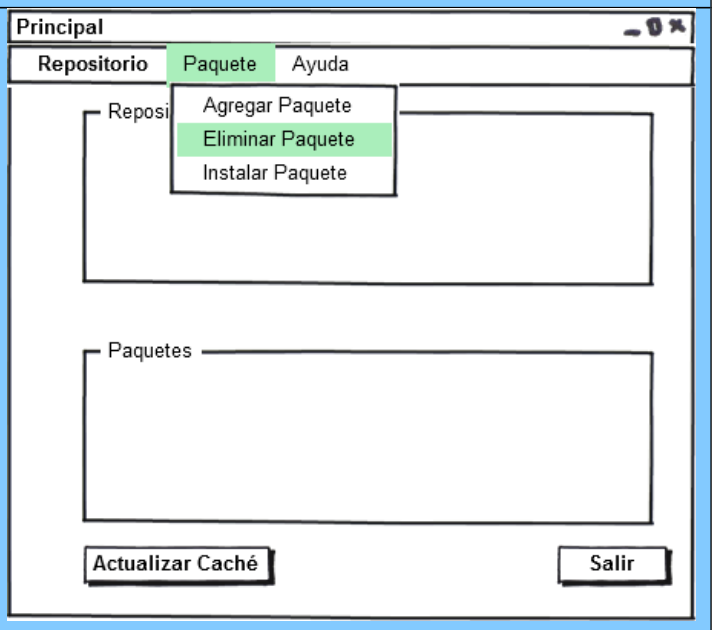
RF4: Eliminar paquete		
Precondiciones	Exista un repositorio local	
Flujo de eventos		
Flujo básico	Eliminar paquete	
Prototipo		
Actor		**Sistema**
1.	1. Inicia el caso de uso.	1. Muestra una lista con los repositorios locales disponibles.
2.	1. Escoge un repositorio.	1. Muestra una lista con los paquetes existentes en ese repositorio.
3.	1. Selecciona los paquetes a eliminar	1. Elimina los paquetes seleccionados. 2. Actualiza el sistema.
4.		1. Fin del caso de uso.
Post-Condiciones		
1.	Queda eliminado el paquete del repositorio local.	
Validaciones		
1.	Exista un repositorio local en el sistema.	
2.	Exista el paquete en el repositorio local.	

Tabla 6: Eliminar paquete

RF5: Configurar repositorio		
Precondiciones	Exista un repositorio local en el sistema.	
Flujo de eventos		
Flujo básico	Configurar repositorio	
Actor		**Sistema**
1.	1. Inicia el caso de uso.	
2.	1. Selecciona un repositorio.	1. Muestra una lista con los paquetes existentes en el repositorio.
3.	1. Selecciona la opción Instalar Paquete.	1. Crea un fichero con el nombre, *nombre_del_repositorio_sources.list*, en la ruta *'/etc/apt/sources.list.d/'* con la configuración del repositorio local. 2. Actualiza el sistema. 3. Ejecuta el Centro de S*oftware.*
4.		1. Fin del caso de uso.
Post-Condiciones		
1.	Queda configurado el sistema para utilizar ese repositorio.	
Validaciones		
1.	El usuario tenga privilegios administrativos.	

Tabla 7: Configurar repositorio

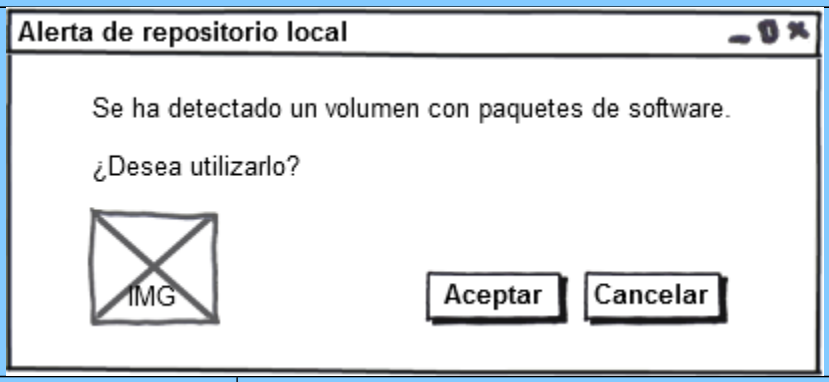
<table>
<tr><th colspan="3">RF6: Detectar dispositivos extraíbles</th></tr>
<tr><td colspan="2">Precondiciones</td><td>Se conecte un dispositivo extraíble.</td></tr>
<tr><td colspan="3">Flujo de eventos</td></tr>
<tr><td colspan="2">Flujo básico</td><td>Detectar dispositivos extraíbles</td></tr>
<tr><td colspan="2">Prototipo</td><td> 
</td></tr>
<tr><td colspan="2">Actor</td><td>Sistema</td></tr>
<tr><td>1.</td><td>1. El usuario inicia el caso de uso.</td><td>1. Revisa el contenido del dispositivo extraíble en busca del fichero hgrp.json.
2. En caso de encontrarlo pide al usuario configurar el repositorio local en el sistema.</td></tr>
<tr><td>2.</td><td>1. Si el usuario confirma que desea utilizarlo.</td><td>1. La información del repositorio local es agregada a la aplicación, permitiendo gestionarlo. Un fichero con la estructura nombre_del_repositorio_sources.list, es agregado a la ruta '/etc/apt/sources.list.d/', permitiendo la instalación de paquetes desde el mismo.
2. En caso contrario termina.</td></tr>
<tr><td>3.</td><td></td><td>1. Fin del caso de uso</td></tr>
<tr><td colspan="3">Post-Condiciones</td></tr>
<tr><td>1.</td><td colspan="2">Queda configurado un nuevo repositorio en el sistema.</td></tr>
<tr><td colspan="3">Validaciones</td></tr>
<tr><td>1.</td><td colspan="2">El usuario tenga privilegios administrativos para realizar las operaciones.</td></tr>
</table>

Tabla 8: Detectar dispositivos extraíbles.

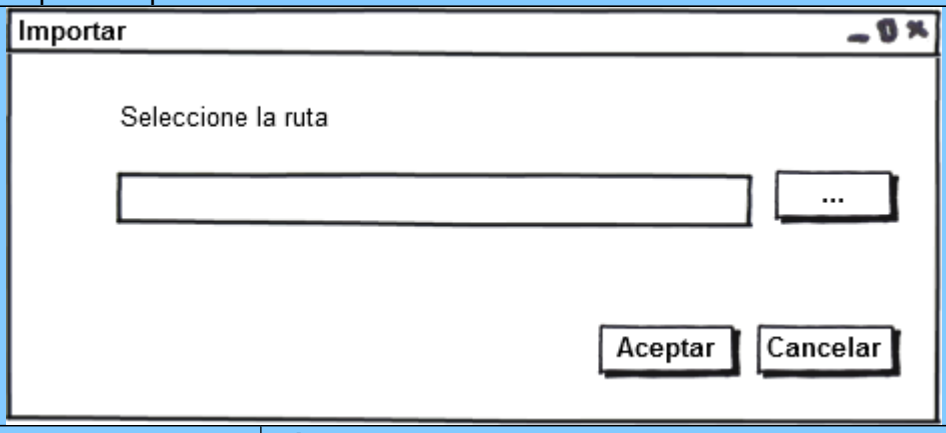
RF7: Importar Repositorio		
Precondiciones	Exista un dispositivo extraíble con al menos un repositorio.	
Flujo de eventos		
Flujo básico	Importar repositorio.	
Prototipo		
Actor		**Sistema**
1.	1. Inicia el caso de uso	1. Pide la ruta del repositorio a importar.
2.	1. Introduce la ruta del repositorio	1. Verifica la validez del repositorio. 2. De ser verdadera la verificación, agrega la información del repositorio al fichero *hgrp.json* 3. Actualiza el sistema.
3.		1. Fin del caso de uso.
Post-Condiciones		
1.	Queda importado el nuevo repositorio en el sistema.	
2.	Queda actualizado el repositorio local.	
Validaciones		
1.	Existe suficiente espacio en el disco duro local para almacenar el repositorio.	

Tabla 9: Importar repositorio

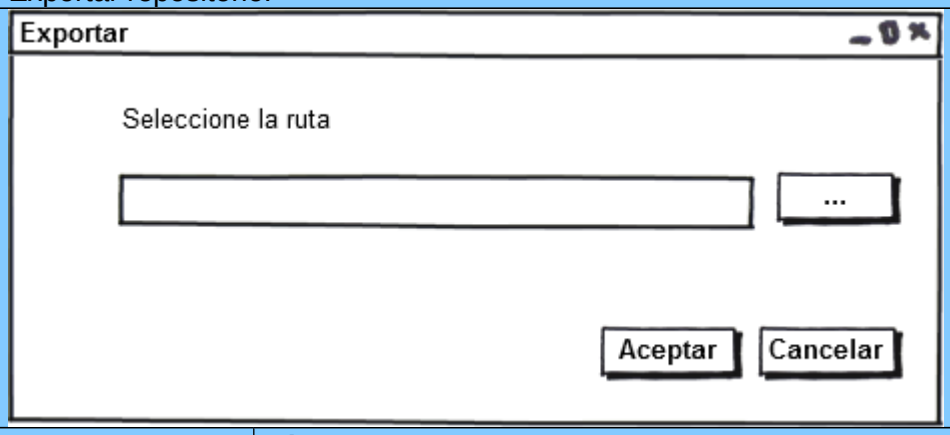
<table>
<tr><th colspan="4">RF8: Exportar Repositorio</th></tr>
<tr><td colspan="2">Precondiciones</td><td colspan="2">Exista un repositorio local.</td></tr>
<tr><td colspan="4">Flujo de eventos</td></tr>
<tr><td colspan="2">Flujo básico</td><td colspan="2">Exportar repositorio.</td></tr>
<tr><td colspan="2">Prototipo</td><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Actor</td><td colspan="2">Sistema</td></tr>
<tr><td>1.</td><td>1. Inicia el caso de uso.</td><td colspan="2">1. Muestra una lista con los repositorios locales.</td></tr>
<tr><td>2.</td><td>1. Selecciona un repositorio.</td><td colspan="2">2. Pide la ruta hacia donde exportar el repositorio</td></tr>
<tr><td>3.</td><td>1. Introduce la ruta.</td><td colspan="2">1. Verifica si hay espacio disponible en el destino.
2. De ser verdadera la verificación exporta el repositorio.
1. Modifica el fichero hgrp.json y agrega la información del nuevo repositorio.
2. Actualiza el sistema.</td></tr>
<tr><td>3.</td><td>1.</td><td colspan="2">1. Fin del caso de uso.</td></tr>
<tr><td colspan="4">Post-Condiciones</td></tr>
<tr><td>1.</td><td colspan="3">Se exporta el repositorio hacia la ruta especificada.</td></tr>
<tr><td>2.</td><td colspan="3">Se añade a la lista de repositorios locales el nuevo repositorio.</td></tr>
<tr><td colspan="4">Validaciones</td></tr>
<tr><td>1.</td><td colspan="3">El destino debe poseer espacio suficiente para almacenar el repositorio.</td></tr>
<tr><td>2.</td><td colspan="3">Debe existir al menos un repositorio local.</td></tr>
</table>

Tabla 10: Exportar repositorio

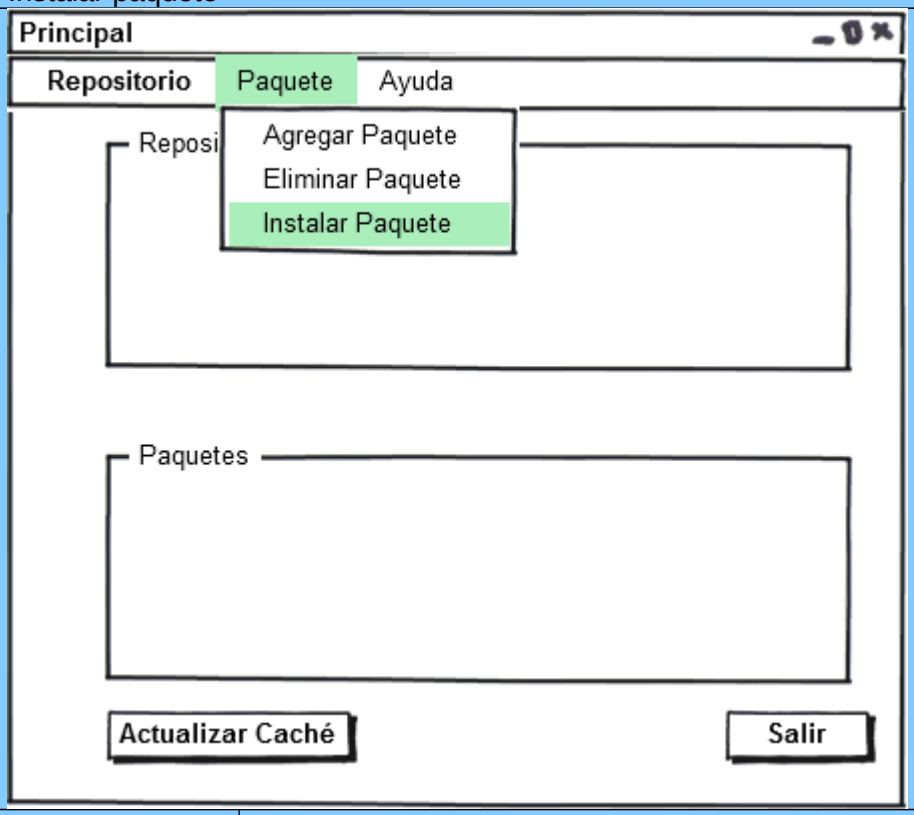
<table>
<tr><th colspan="3">RF9: Instalar paquete</th></tr>
<tr><td>Precondiciones</td><td colspan="2">Exista al menos un paquete en un repositorio local</td></tr>
<tr><td colspan="3">Flujo de eventos</td></tr>
<tr><td>Flujo básico</td><td colspan="2">Instalar paquete</td></tr>
<tr><td>Prototipo</td><td colspan="2"> 
</td></tr>
<tr><td colspan="2">Actor</td><td>Sistema</td></tr>
<tr><td>1.</td><td>1. Inicia el caso de uso.</td><td></td></tr>
<tr><td>2.</td><td>1. Selecciona un repositorio de la lista de repositorios locales disponibles.
2. Selecciona un paquete.
3. Selecciona la acción Instalar Paquete.</td><td>1. Crea un fichero con el nombre, nombre_del_repositorio_sources.list, en la ruta '/etc/apt/sources.list.d/' con la configuración del repositorio local.
2. Actualiza el sistema.
3. Ejecuta el Centro de Software.</td></tr>
<tr><td>3.</td><td></td><td>1. Fin del caso de uso.</td></tr>
<tr><td colspan="3">Post-Condiciones</td></tr>
<tr><td>1.</td><td colspan="2">Se ejecuta el Centro de Software.</td></tr>
<tr><td colspan="3">Validaciones</td></tr>
<tr><td>1.</td><td colspan="2">El paquete este descargado con todas sus dependencias.</td></tr>
<tr><td>2.</td><td colspan="2">Pertenezca a un repositorio local.</td></tr>
<tr><td>3.</td><td colspan="2">Este correctamente configurado el sistema para trabajar con el repositorio local.</td></tr>
</table>

Tabla 11: Instalar paquete

2.6 Requisitos no funcionales

A continuación se muestran los requisitos no funcionales clasificados por categorías:

Usabilidad

El usuario final deberá poseer privilegios administrativos en caso que desee utilizar la funcionalidad de instalar.

Compatibilidad

- El sistema estará integrado con la distribución de GNU/Linux Nova.
- Cumplirá los estándares de comunicación con el Centro de *Software* para consolidar las funcionalidades de un sistema gestor de repositorios.

Restricciones de diseño

- Las normas de codificación se van a llevar a cabo a partir de lo definido por la guía de estilo para el código Python PEP8.
- La arquitectura se va a describir bajo el patrón arquitectónico N-Capas (3-Capas).

Eficiencia

- La memoria RAM de la estación de trabajo debe tener una capacidad mínima de un 1GB.

Interfaz

- Interfaz sencilla e intuitiva.

2.7 Diagrama de Secuencia (DS)

Un diagrama de secuencia muestra las interacciones que existen entre los objetos, ordenadas en secuencia temporal durante la interacción de agentes externos desarrollados en un escenario concreto. A continuación se ejemplifican los Diagramas de Secuencia para los Casos de Uso más significativos.

Gestionar repositorio:

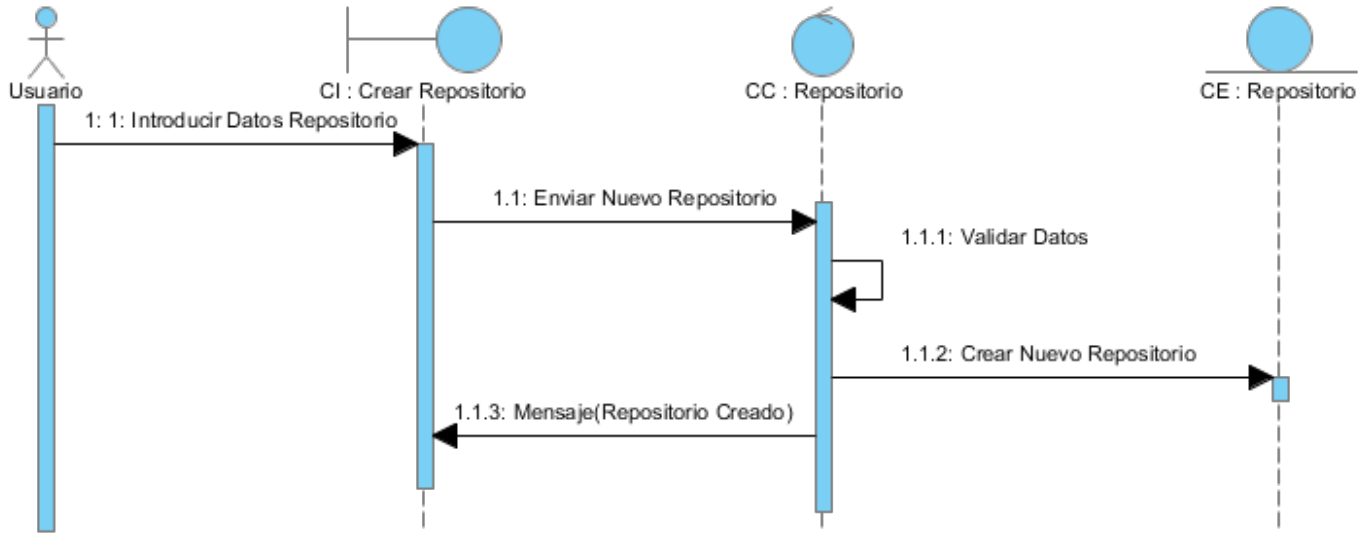


Ilustración 3: DS_Crear repositorio

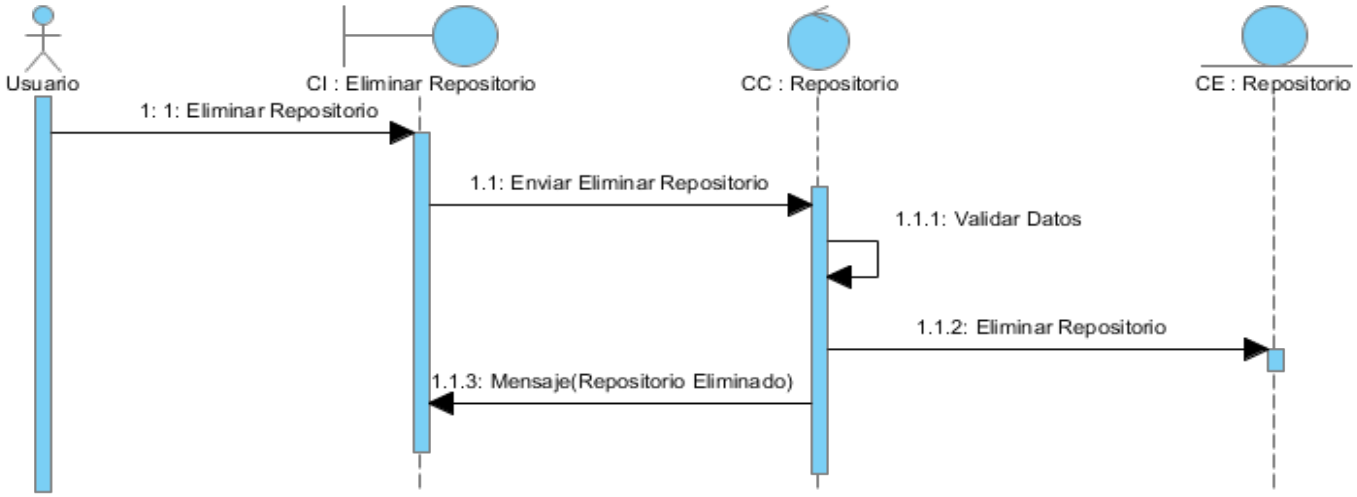


Ilustración 4: DS_Eliminar repositorio

Gestionar paquete:

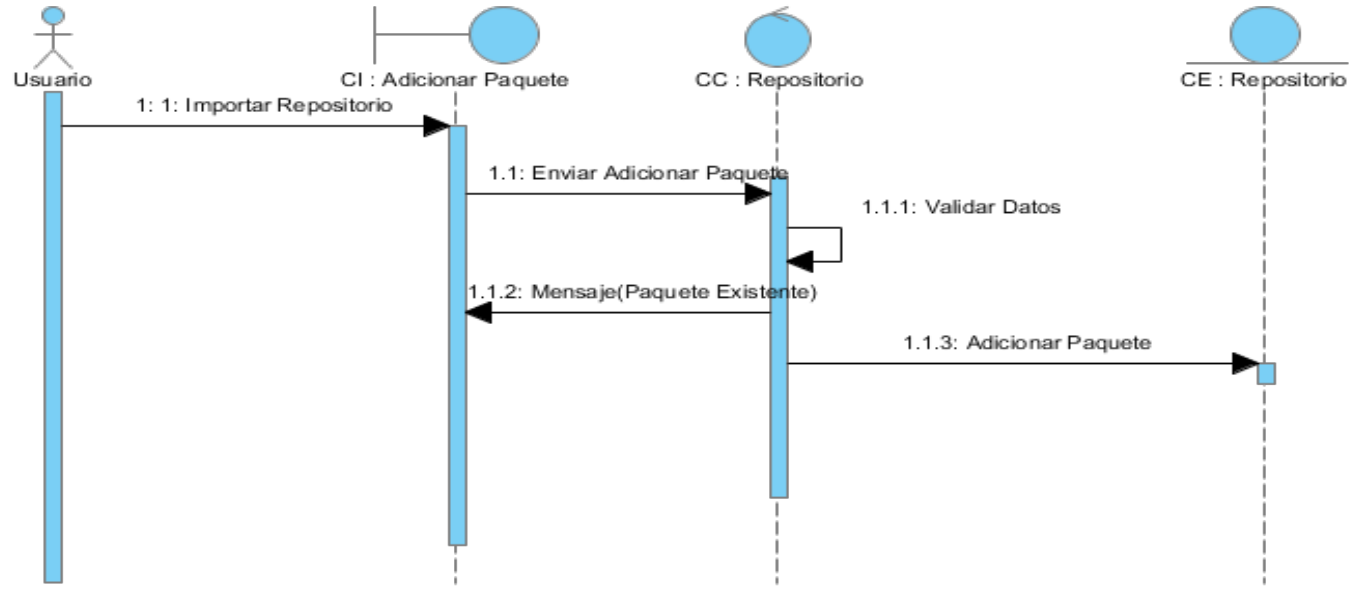


Ilustración 5: DS_Agregar paquete

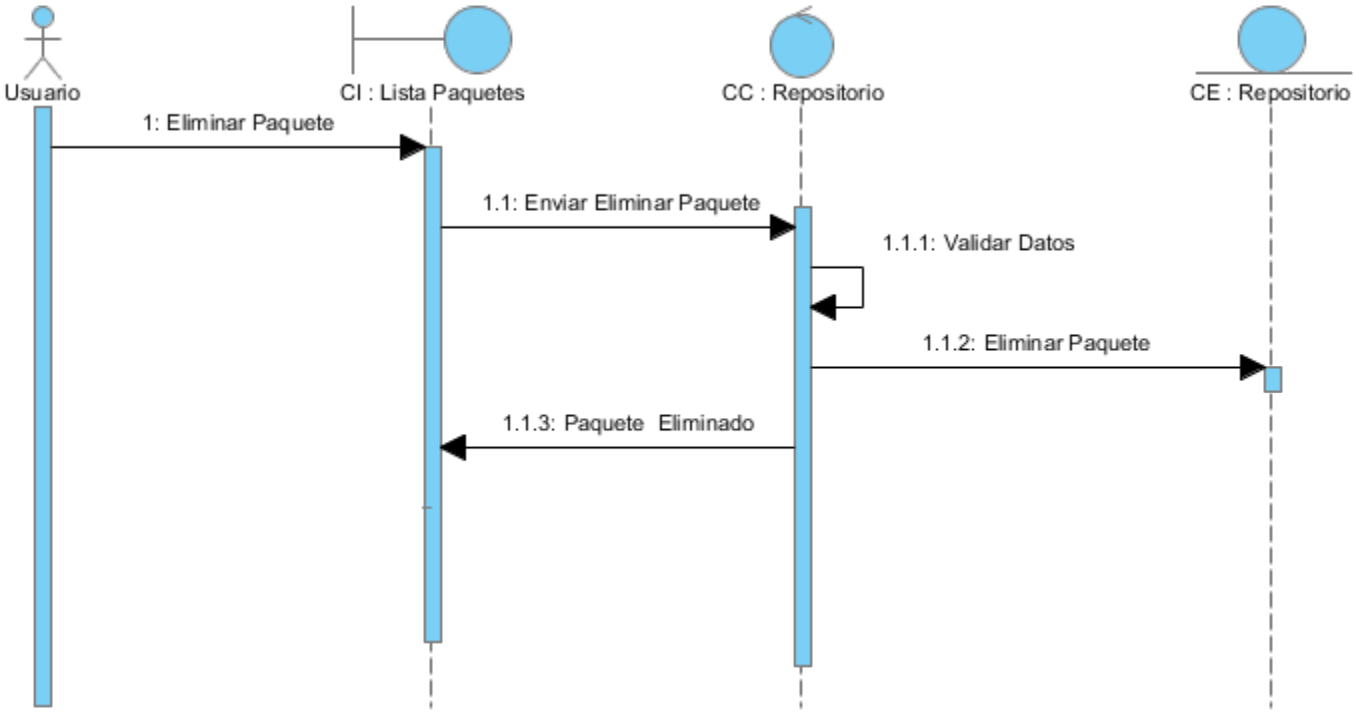


Ilustración 6: DS_Eliminar paquete

Capítulo 2: Análisis y diseño de la solución propuesta

2.8 Descripción de la arquitectura

La arquitectura de *software* es una representación que permite que un ingeniero del *software* analice la efectividad del diseño para cumplir con los requisitos establecidos, considere opciones arquitectónicas en una etapa en que aún resulta relativamente fácil hacer cambios al diseño y reduzca los riesgos asociados con la construcción del *software* (14). Está compuesta por elementos arquitectónicos fundamentales en el diseño de un *software*, estos son: Patrón Arquitectónico y Patrón de Diseño.

2.8.1 Estilo arquitectónico

El estilo arquitectónico en capas se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva en los problemas a resolver. Los roles se indican en el tipo y la forma de la interacción en otras capas y las funcionalidades que implementan (15).

Beneficios:

- Abstracción: los cambios se realizan a alto nivel y se puede incrementar o reducir el nivel de abstracción que se usa en cada capa del modelo.
- Aislamiento: se pueden realizar actualizaciones en el interior de las capas sin que esto afecte al resto del sistema.
- Rendimiento: distribuyendo las capas en distintos niveles físicos se puede mejorar la escalabilidad, la tolerancia a fallos y el rendimiento.
- Testeabilidad: cada capa tiene una interfaz bien definida sobre la que realizar las pruebas y la habilidad de cambiar entre diferentes implementaciones de una capa.
- Independencia: elimina la necesidad de considerar el *hardware* y el despliegue así como las dependencias con interfaces externas.

Se establecieron tres capas para separar cada una de acuerdo con sus responsabilidades, la Capa de Presentación, la Capa Controladora, y la Capa de Acceso a Datos.

A través de la Capa de Presentación el usuario interactúa con el sistema mediante las diferentes interfaces visuales, la principal función de esta capa es mostrar información al usuario y recibir

los datos introducidos para su posterior validación. La Capa Controladora procesa los valores introducidos por el usuario y hace las llamadas a los métodos contenidos en la Capa de Acceso a Datos, esta capa hace una petición a los datos contenidos en la fuente de actualización y le devuelve a la capa superior esos valores procesados para posteriormente ser mostrados al usuario.

Como peculiaridad de este diseño se tiene la carencia de una base de datos propia del sistema, por lo que el acceso a datos en realidad serían peticiones a una fuente de actualización para listar los paquetes disponibles en la misma. Para mayor comprensión ver Anexo 13.

2.8.2 Patrones de diseño

GRASP es un acrónimo de *General Responsibility Assigment Software Patterns* (patrones generales de *software* para asignar responsabilidades). Son utilizados para describir los principios fundamentales del diseño y la asignación de responsabilidades (16).

Experto: Consiste en asignar la responsabilidad a la clase que tiene la información necesaria para realizarla. La clase localRepo es la encargada de descargar los paquetes y sus dependencias.

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. La clase *ControllerMainWindow* crea una instancia de la clase *Repositorio*, lo que permite la modificación de los atributos de los repositorios locales en tiempo de ejecución.

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.

Las clases que se encuentran ubicadas en la capa de presentación y las clases ubicadas en la capa de acceso a datos tienen bastante independencia entre si, haciendo muy fácil el proceso de modificación de código sin afectar el funcionamiento de la otra.

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 tra-

bajo. 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. La clase *localRepo* es la encargada de descargar los paquetes de la fuente de actualización, y devolver la información a las capas superiores para ser procesada y mostrada al usuario, solo esta clase tiene las funcionalidades necesarias para realizar esta operación.

2.9 Conclusiones parciales

En este capítulo se mostró una visión general de lo que será la solución del sistema, precisándose los requisitos a cumplirse, desglosados en Casos de Uso y estableciendo un nivel de prioridad para cada uno. Además se modeló el diagrama de Casos de Uso del sistema, el diagrama de Clases, el modelo de Dominio y los diagramas de Secuencia. También se definió el estilo arquitectónico y los patrones de diseño de la aplicación para así facilitar la implementación de la solución propuesta.

Capítulo 3. Implementación y prueba al sistema

3.1 Introducción

En el siguiente capítulo se muestra la descripción del diseño de las pruebas realizadas a la herramienta HGRP y los resultados obtenidos durante cada iteración. Se creó el Diagrama de Componentes, mostrando la relación entre cada uno. Además se explican de manera breve los principales métodos implementados en la solución propuesta.

3.2 Diagrama de Componentes

A continuación se muestra, en la ilustración 7, el diagrama de componentes propuesto para el sistema HGRP, donde:

- **Principal.py**: contiene las clases necesarias para la interacción visual del usuario con el sistema y es a su vez la que controla todo el proceso de configuración, interactuando con los componentes LocalRepo.py y Repositorio.py.
- **LocalRepo.py**: se encarga de la descarga de los paquetes y sus dependencias existentes en la fuente de actualización, así como el encargado de la creación de la caché para el sistema mediante el método downloadCache(self).
- **Repositorio.py**: guarda la información de cada repositorio local creado en el sistema, propiciándole a la clase Principal.py el fácil manejo de los datos en tiempo de ejecución antes de guardar el estado final del sistema en el fichero *hgrp.json*.
- **Subsistema Detección Automática**: informa a la clase controladora si se ha conectado un nuevo dispositivo con un repositorio válido para la aplicación y de ser así, configura el sistema para usarlo, esto se realiza mediante el subsistema Validaciones y Configuración Automática respectivamente.
- **Subsistema Relocalización de Repositorios**: provee las vistas necesarias para importar y exportar un repositorio, mediante los componentes Importar.py y Exportar.py respectivamente.

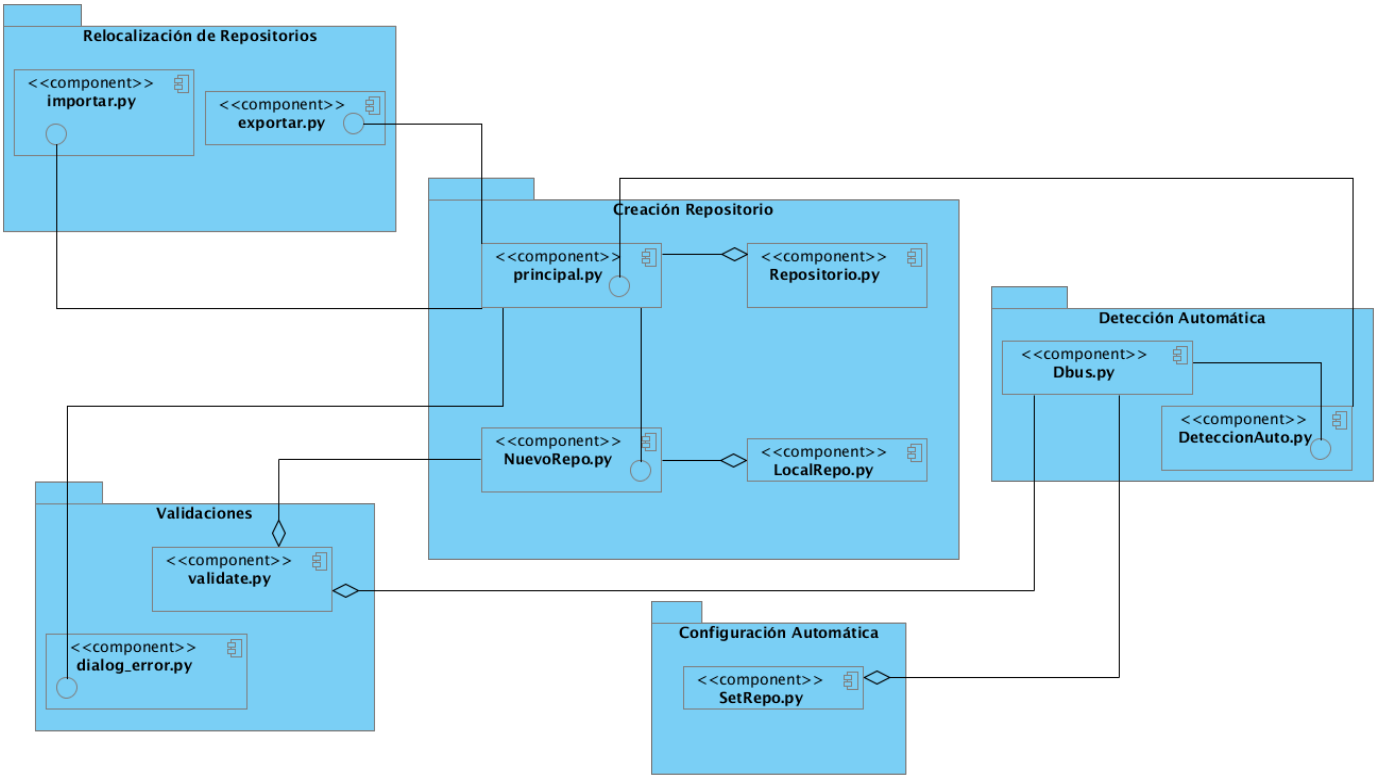


Ilustración 7: Diagrama de Componentes

3.3 Pruebas de *Software*

El desarrollo de cada sistema informático debe pasar por un conjunto de pruebas con el objetivo de comprobar si cumple con los requisitos definidos inicialmente y además validar su buen funcionamiento. Para el caso de HGRP se requieren realizar pruebas que demuestren que cada función es plenamente operacional, estas se denominan pruebas de caja negra. Otra variante es cuando se conoce el funcionamiento interno del producto, se aplican pruebas para asegurarse de que todas las piezas encajan, estas son las pruebas de caja blanca (14). Durante todo el proceso de desarrollo se le hicieron al sistema pruebas unitarias, las que confirmaron la efectividad de las pruebas de caja blanca. A continuación se muestran los resultados arrojados por las pruebas de aceptación.

3.3.1 Pruebas de Aceptación

Dentro de los tipos de pruebas de caja negra se encuentran las Pruebas de Aceptación. Estas pruebas determinan la aceptación o rechazo del sistema desarrollado por parte del cliente,

además sirven para que el usuario pueda validar si el producto final se ajusta a los requisitos fijados, es decir, si el producto está listo para ser implantado para el uso operativo en el entorno del usuario. A continuación se definen los Casos de Pruebas de Aceptación (CPA) para los CU1 y CU2**.**

Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-01-01	**Nombre Caso de Uso**: Gestionar repositorio
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se crea un repositorio local en la ruta especificada por el usuario.	
Condiciones de Ejecución: Tener acceso a una fuente de actualización.	
Entrada / Pasos de ejecución: **1.** Selecciona el menú Agregar Repositorio. **2.** Introduce los datos requeridos. **3.** Selecciona el botón Actualizar. **4.** Selecciona los paquetes deseados. **5.** Selecciona el botón Crear.	
Resultado Esperado: Se muestra en la interfaz principal el nombre del repositorio local, al seleccionarlo se muestra la lista de los paquetes que contiene.	
Evaluación de la Prueba: Satisfactoria	

Tabla 12: CPA_01: Gestionar repositorio

Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-01-02	**Nombre Caso de Uso**: Gestionar repositorio
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se elimina un repositorio local con todos sus paquetes.	
Condiciones de Ejecución: Deben existir el repositorio.	
Entrada / Pasos de ejecución: **1.** Se selecciona el repositorio deseado en la lista de repositorios disponibles. **2.** Se pulsa el menú Eliminar Repositorio.	
Resultado Esperado: El sistema muestra la información actualizada de los repositorios locales existentes en el sistema.	
Evaluación de la Prueba: Satisfactoria	

Tabla 13: CPA_02: Gestionar repositorio

Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-02-01	**Nombre Caso de Uso:** Gestionar paquete
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se adicionan paquetes al repositorio local.	
Condiciones de Ejecución: Debe existir al menos un repositorio local.	
Entrada / Pasos de ejecución: **1.** Se selecciona el repositorio local deseado en la lista de repositorios disponibles. **2.** Se selecciona el menú Adicionar Paquete. **3.** Se muestra la interfaz para agregar un nuevo repositorio con los datos previos de ese repositorio. **4.** Se configuran los parámetros arquitectura y distribución. **5.** Se pulsa el botón Actualizar. **6.** Se muestra la lista de paquetes disponibles para agregar al repositorio. **7.** Se seleccionan los paquetes deseados y se pulsa el botón Crear.	
Resultado Esperado: El sistema muestra los repositorios actualizados.	
Evaluación de la Prueba: Satisfactoria	

Tabla 14: CPA_01: Gestionar paquete

Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-02-01	**Nombre Caso de Uso:** Gestionar paquete
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza.	
Descripción de la Prueba: Se eliminan paquetes del repositorio local.	
Condiciones de Ejecución: Debe existir al menos un repositorio local.	
Entrada / Pasos de ejecución: **1.** Se selecciona el repositorio local deseado en la lista de repositorios disponibles. **2.** Se selecciona el paquete que se desea eliminar. **3.** Se selecciona el menú Eliminar Paquete.	
Resultado Esperado: El sistema muestra los repositorios actualizados.	
Evaluación de la Prueba: Satisfactoria	

Tabla 15: CPA_02: Gestionar paquete

Resultados de las Pruebas de Aceptación

A continuación se identifican los resultados arrojados según las pruebas de Aceptación realizadas a la aplicación. Se realizaron 3 iteraciones de pruebas, una primera iteración para los CP1, CP2 y CP3, una segunda para los CP4, CP5 y CP6 y una tercera donde se realizó una prueba a los CP7, CP8 y CP9, corrigiendo las No Conformidades (NC) detectadas.

- En la primera iteración se obtuvo como resultado un total de 10 No Conformidades (NC), de ellas 5 Significativas (S), 2 No Significativas (NS) y 3 Recomendaciones (R).
- La segunda iteración de pruebas arrojó como resultado 15 NC, de ellas 9 S, 4 NS y 2 R.
- La tercera iteración no arrojó ninguna NC.

La siguiente gráfica muestra los resultados obtenidos para cada una de las iteraciones.

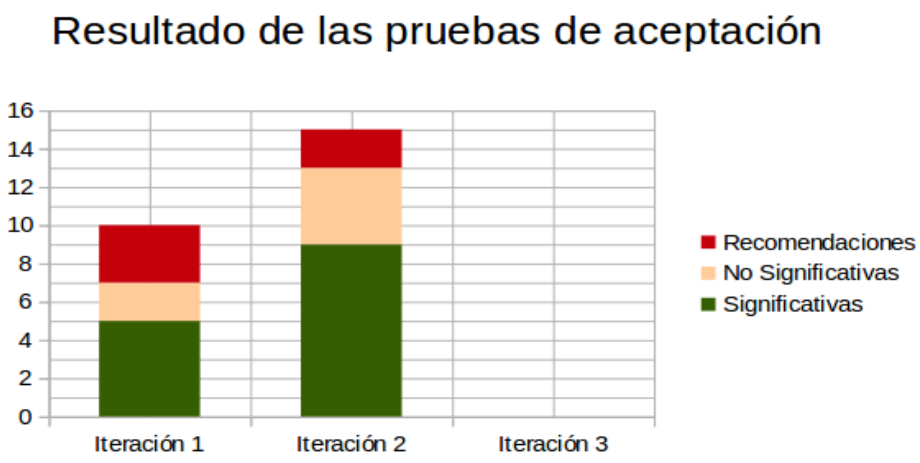


Ilustración 8: Resultados de las Pruebas de Aceptación

3.4 Descripción de los principales métodos implementados.

Para lograr un correcto funcionamiento de la aplicación se implementaron los siguientes métodos:

- DownloadCache(): encargado de descargar la caché del sistema.
- CachedPackages(): devuelve la lista de paquetes disponibles en una fuente de actualización determina.
- ListDep(package): devuelve la lista de dependencias de un paquete determinado.
- Download(package): método que descarga un paquete determinado.
- DownloadList(list): descarga una lista de paquetes.
- DownloadAll(package): es el encargado de hacer las llamadas a todos los métodos anteriores en el orden pertinente, es quien lleva a cabo todo el proceso de descarga de paquetes con sus dependencias.
- ReadyRepo(): encargado de crear los repositorios locales con la estructura clásica.

Método DownloadCache

Este método se implementa con el objetivo de descargar la caché para una fuente de actualización y hacer la búsqueda de paquetes una sola vez. Primero se verifica que no exista la caché en el sistema, de ser así se procede a la búsqueda de todos los ficheros *Release* en la URL solicitada. Si no se encuentra ningún fichero se lanza una excepción de repositorio no válido. En caso contrario se leen los campos, *CodeName* y *Components* para proceder a la descarga de los ficheros *Packages*, que contienen la información de los paquetes existentes en la fuente de actualización. De no existir la caché en el sistema se procede a su descarga.

Método CachedPackages

Busca la existencia de la caché en el sistema para la fuente de actualización escogida. Obtiene la lista de paquetes disponibles y la devuelve para ser mostrada al usuario.

Método ListDep

Hace una llamada al método *getDependencies*, encargado de devolver la lista de dependencias para un paquete dado, en caso de que alguna dependencia, tenga a su vez, una lista de dependencias, procede a su búsqueda hasta agotar toda la lista.

Método Download

Hace una llamada al método *getFilename* para obtener la ruta del paquete en la fuente de actualización y proceder a su descarga. Descarga el paquete, dentro de la carpeta escogida por el usuario, manteniendo su ruta oficial a partir de la carpeta */pool.* En caso de no existir la ruta la crea.

Método DownloadList

Recibe una lista de paquetes a descargar y para cada paquete en la lista hace una llamada al método *DownloadAll*.

Método DownloadAll

Recibe el nombre de un paquete y hace una llamada al método *Download* tomando como parámetro la salida del método *getPackage* que sería el nombre completo del paquete a

descargar. A continuación se procede a buscar todas las dependencias de ese paquete para su posterior descarga mediante el método *Download*, en caso de no existir la dependencia se procede a analizar la próxima en la lista.

Método ReadyRepo

Hace una llamada al comando *dpkg-scanpackages* pasandóle como parámetro el *pool* del repositorio local ya creado, este método genera un fichero *Package* para cada uno de los componentes[12], arquitecturas[13] y *codename*[14] de los repositorios descargados. Estos ficheros serán utilizados para validar el posterior proceso de instalación de paquetes. Con el comando *gzip* se generan los compactados, con extensión *gz,* de cada uno de los ficheros *Packages.*

3.5 Conclusiones Parciales

A lo largo de este capítulo se realizó la descripción de los casos de prueba basados en requisitos utilizados durante las iteraciones de pruebas a la Herramienta de Gestión de Repositorios Locales y Portables. Para realizar la verificación del sistema se definió utilizar las técnicas de caja blanca y caja negra, para darle un mejor acabado al producto que se deseaba obtener siguiendo los requerimientos del sistema. Se diseñó el diagrama de Componentes para un mejor entendimiento del funcionamiento de la solución propuesta. Se realizó una breve descripción de los principales métodos encargados de la descarga de paquetes con el objetivo de lograr una mejor comprensión de este proceso. Se detectaron y corrigieron una serie de no conformidades gracias a la realización del proceso de pruebas, validando así el correcto funcionamiento del sistema.

12 Para el caso de la distribución GNU/Linux Nova serían principal y extendido.

13 Arquitecturas para las que se descargaron los paquetes, i386 o amd64.

14 Para el caso de la distribución GNU/Linux Nova serían 2011 y 2013.

Conclusiones

Con la culminación del presente trabajo de diploma se cumplieron cada uno de los objetivos trazados, distinguiéndose de manera general los siguientes aspectos:

1. El estudio de sistemas homólogos de herramientas para la gestión de repositorios permitió definir una serie de características para la solución propuesta
2. El desarrollo de la solución propuesta permitió prescindir de una conexión de red para gestionar los repositorios de *software* de manera local, garantizando además su portabilidad.
3. La herramienta permitió facilitar el proceso de instalación de paquetes provenientes de repositorios locales, mediante la automatización del proceso de configuración de los repositorios de Nova, utilizando las funcionalidades del Centro de *Software*.
4. Las pruebas realizadas al producto implementado garantizaron la usabilidad del mismo y evidenciaron que cumple con los requerimientos definidos al inicio de la investigación, estas permitieron la corrección de las no conformidades encontradas.

Recomendaciones

1. Modificar la herramienta con el objetivo de que permita el uso de otras arquitecturas de *software* como *ARM*.
2. Incluir filtros por categorías para organizar las aplicaciones a la hora de mostrarlas al usuario para así facilitar su búsqueda.
3. Establecer una prioridad para los repositorios locales, una vez que se decida instalar aplicaciones de alguno de ellos que se encuentre configurado en el sistema.
4. Permitir la compresión del repositorio en formato ISO y guardarlo en un soporte de almacenamiento.

Referencias bibliográficas

1. Repositorios en Ubuntu Linux, que son y para qué sirven. *BajoLinux* [online]. 3 December 2013. [Accessed 3 February 2014]. Available from: http://bajolinux.com/repositorios-ubuntu-linux-que-son-para-que-sirven/

2. *Ubuntu Comunity. Instalar software - doc.ubuntu-es.* [online]. 25 April 2012. Available from: http://doc.ubuntu-es.org/Instalar_software#.C2.BFQu.C3.A9_es_un_paquete.3F

3. PÉREZ, Adrián Lázaro Lara. *Sistema para la creación de repositorios personalizados en línea* [online]. (Ingeniero en Ciencias Informáticas). La Hababa : Universidad de las Ciencias Informáticas, 2012. Available from: http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=11685

4. *APTonCD. User's Manual* [online]. [Accessed 5 February 2014]. Available from: http://aptoncd.sourceforge.net/doc-manual.html

5. LÓPEZ, José María. Gestor de paquetes para equipos sin Internet. *Softonic* [online]. [Accessed 25 March 2014]. Available from: http://keryx.softonic.com/linux

6. *OpenSuse team. OpenSuse Project. OSP.* [online]. [Accessed 3 March 2014]. Available from: http://en.opensuse.org/Yast

7. THOMAS, Matthew Paul. SoftwareCenter. *ubuntu wiki* [online]. 29 August 2005. [Accessed 10 January 2014]. Available from: https://wiki.ubuntu.com/SoftwareCenter

8. *Introduction to Open UP* [online]. Available from: http://epf.eclipse.org/openup/

9. ROSSUM, Guido. *El tutorial de Python.* [online]. 2009. Available from: http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf

10. HERRERA, Dariem Pérez. Comunicando aplicaciones con D Bus (Parte 1). *Humanos* [online]. 4 November 2013. Available from: http://humanos.uci.cu/2013/11/humancode-comunicando-aplicaciones-con-d-bus-parte-i/

Referencias bibliográficas

11. KLODE, Julian Andres. *python-apt v0.8.0~exp4 documentation* [online]. 29 April 2011. [Accessed 20 March 2014]. Available from: http://apt.alioth.debian.org/python-apt-doc/library/index.html

12. GUTIÉRREZ, David González. *Tutorial de Qt4 Designer y QDevelop. FIB-UPC.* 2009.

13. SIERRA, María. *Trabajando con Visual Paradigm for UML.* [online]. 2005. Available from: http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf

14. PRESSMAN, Roger S. El proceso de investigación científica. In : *Ingeniería del Software.Un enfoque práctico. La Habana : Félix Varela, 2005. p. 418 Cap. 14.*

15. LLORENTE, César de la Torre, CASTRO, Unai Zorrilla, BARROS, Miguel Angel Ramos and NELSON, Javier Calvarro. Guía de Arquitectura N Capas orientada al Dominio con. NET 4.0. 4. España, 2010. ISBN 978-84-936696-3-8.

16. LARMAN, Craig. *UML y Patrones. Introducción al análisis orientado a objeto*. 2004. ISBN 970-17-0261-1.

Bibliografía consultada

1. *APTonCD. User's Manual* [online]. [Accessed 5 February 2014]. Available from: http://aptoncd.sourceforge.net/doc-manual.html

2. *BAHIT, Eugenia. POO y MVC en PHP. El paradigma de la Programación Orientada a Objetos en PHP y el patrón de arquitectura de Software MVC* [online]. 2011. [Accessed 27 March 2014]. Available from: http://eugeniabahit.blogspot.com/2011/07/poo-y-mvc-en-php.htmlpdf

3. CUÉLLAR, Jose, [no date], Estilos arquitecturales en el diseño de un sistema. *Jose Cuéllar.net* [online]. Available from: http://www.josecuellar.net/arquitectura-de-software/estilos-arquitecturales-en-el-diseno-de-un-sistema/

4. Definición de Arquitectura Software-Significado y definición de Arquitectura Software, [no date]. [online], [Accessed 9 December 2013]. Available from: http://www.mastermagazine.info/termino/3916.php

5. GARCÍA, Marisol Lara. *Ingeniería de Software II*. 2013.

6. GUTIÉRREZ, David González. *Tutorial de Qt4 Designer y QDevelop. FIB-UPC.* 2009.

7. HERRERA, Dariem Pérez. Comunicando aplicaciones con D Bus (Parte 1). *Humanos* [online]. 4 November 2013. Available from: http://humanos.uci.cu/2013/11/humancode-comunicando-aplicaciones-con-d-bus-parte-i/

8. *Introduction to Open UP* [online]. Available from: http://epf.eclipse.org/openup/

9. JAQUELINE, Laura, [no date], PATRONES GRASP. *Prezi* [online]. [Accessed 5 December 2013]. Available from: http://prezi.com/di0m87zy1lpo/patrones-grasp/

10. KLODE, Julian Andres. *python-apt v0.8.0~exp4 documentation* [online]. 29 April 2011. [Accessed 20 March 2014]. Available from: http://apt.alioth.debian.org/python-apt-doc/library/index.html

Bibliografía consultada

11. LARMAN, Craig. *UML y Patrones. Introducción al análisis orientado a objeto*. 2004. ISBN 970-17-0261-1.

12. LÓPEZ, José María. Gestor de paquetes para equipos sin Internet. *Softonic* [online]. [Accessed 25 March 2014]. Available from: http://keryx.softonic.com/linux

13. Modelo de Implementación: Diagramas de Componentes y Despliegue. Ingeniería del Software (3o I.T.I.S., I.T.I.G.) Módulo 2. Tema 12: Modelo de Implementación. Febrero 21, 2012.

14. *OpenSuse team. OpenSuse Project. OSP.* [online]. [Accessed 3 March 2014]. Available from: http://en.opensuse.org/Yast

15. Patrones Arquitectónicos | Ingenio DS, [no date]. [online], [Accessed 5 November 2013]. Available from: http://ingeniods.wordpress.com/2013/09/16/patrones-arquitectonicos/

16. PÉREZ, Adrián Lázaro Lara. *Sistema para la creación de repositorios personalizados en línea* [online]. (Ingeniero en Ciencias Informáticas). La Hababa : Universidad de las Ciencias Informáticas, 2012. Available from: http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=11685

17. PRESSMAN, Roger S. El proceso de investigación científica. In : *Ingeniería del Software.Un enfoque práctico. La Habana : Félix Varela, 2005. p. 418 Cap. 14.*

18. Repositorios en Ubuntu Linux, que son y para que sirven. *BajoLinux* [online]. 3 December 2013. [Accessed 3 February 2014]. Available from: http://bajolinux.com/repositorios-ubuntu-linux-que-son-para-que-sirven/

19. ROSSUM, Guido Van. *Guía de estilo para el código Python – PEP 8 en Español* [online]. 2013. Available from: www.recursospython.com

20. SIERRA, María. *Trabajando con Visual Paradigm for UML.* [online]. 2005. Available from: http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf

Bibliografía consultada

21. THOMAS, Matthew Paul. SoftwareCenter. *ubuntu wiki* [online]. 29 August 2005. [Accessed 10 January 2014]. Available from: https://wiki.ubuntu.com/SoftwareCenter

22. Tutorial de UML, [no date]. [online], [Accessed 20 January 2014]. Available from: http://users.dcc.uchile.cl/~psalinas/uml/introduccion.html

23. *Ubuntu Comunity. Instalar software - doc.ubuntu-es.* [online]. 25 April 2012. Available from: http://doc.ubuntu-es.org/Instalar_software#.C2.BFQu.C3.A9_es_un_paquete.3F

24. Unified Modeling Language (UML), [no date]. [online], [Accessed 20 January 2014]. Available from: http://www.uml.org/

Anexos

Caso de Prueba de Aceptación:

	Caso de Prueba de Aceptación
Código Caso de Prueba: HGRP-05-01	**Nombre Caso de Uso:** Configurar el repositorio
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: El sistema será capaz de configurar el repositorio de manera automática, agregando un fichero con la ruta del repositorio en el directorio *'/etc/apt/sources.list.d/'*	
Condiciones de Ejecución: Se seleccione la opción Instalar Paquete en el menú Archivo o se detecta un repositorio en un dispositivo extraíble.	
Entrada / Pasos de ejecución:	
Resultado Esperado: Queda configurado el sistema para utilizar el nuevo repositorio.	
Evaluación de la Prueba: Satisfactoria	

Anexo 1: CPA_01: Configurar repositorio

	Caso de Prueba de Aceptación
Código Caso de Prueba: HGRP-06-01	**Nombre Caso de Uso:** Detectar dispositivo extraíble.
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se iniciará la aplicación una vez detectado un dispositivo extraíble con la estructura de un repositorio.	
Condiciones de Ejecución: Se conecte un dispositivo extraíble con un repositorio creado previamente por la aplicación.	
Entrada / Pasos de ejecución: **1.** En caso de seleccionar que se desea utilizar el repositorio detectado, el sistema crea un fichero en la ruta '/etc/apt/sources.list.d/' con la configuración del repositorio necesaria para ser usado por el sistema. **2.** En caso contrario no se hace nada.	
Resultado Esperado: Queda configurado el sistema para utilizar el nuevo repositorio.	
Evaluación de la Prueba: Satisfactoria	

Anexo 2: CPA_01: Detectar dispositivo extraíble.

Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-07-01	**Nombre Caso de Uso:** Importar repositorio
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se importa hacia la computadora un repositorio existente en un dispositivo extraíble.	
Condiciones de Ejecución: Debe existir un repositorio en el dispositivo extraíble creado previamente por la aplicación.	
Entrada / Pasos de ejecución: 1. Se selecciona el repositorio a importar en la lista de repositorios disponibles configurados en el sistema. 2. En el menú Archivo se selecciona la opción Importar Repositorio. 3. Se pulsa el botón ... y se selecciona la ruta destino del repositorio. 4. Se pulsa el botón Aceptar.	
Resultado Esperado: El sistema muestra la información actualizada de los repositorios locales existentes en el sistema.	
Evaluación de la Prueba: Satisfactoria	

Anexo 3: CPA_02: Importar repositorio

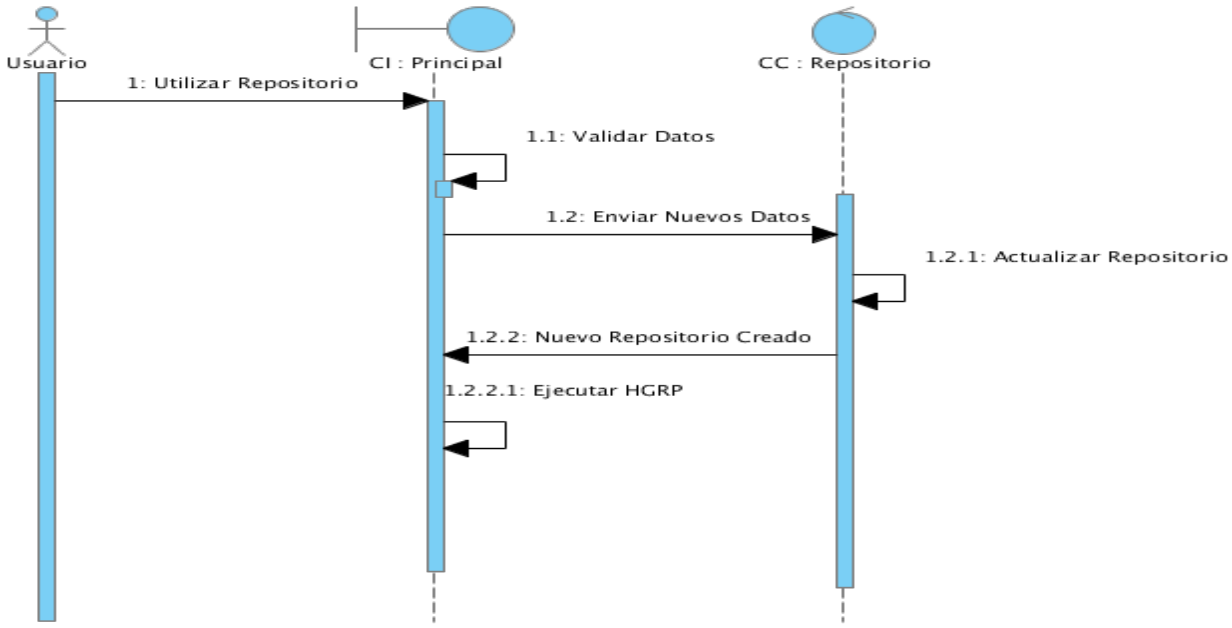
Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-08-01	**Nombre Caso de Uso:** Exportar repositorio.
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se exporta un repositorio hacia una ruta especificada por el usuario.	
Condiciones de Ejecución: Deben existir al menos un repositorio configurado en el sistema	
Entrada / Pasos de ejecución: 1. Se selecciona un repositorio de la lista de repositorios locales disponibles. 2. En el menú Archivo se selecciona la opción Exportar Repositorio. 3. Se pulsa el botón ... y se selecciona la ruta destino del repositorio. 4. Se pulsa el botón Aceptar.	
Resultado Esperado: El sistema muestra la información actualizada de los repositorios locales existentes en el sistema.	
Evaluación de la Prueba: Satisfactoria	

Anexo 4: CPA_02: Exportar repositorio

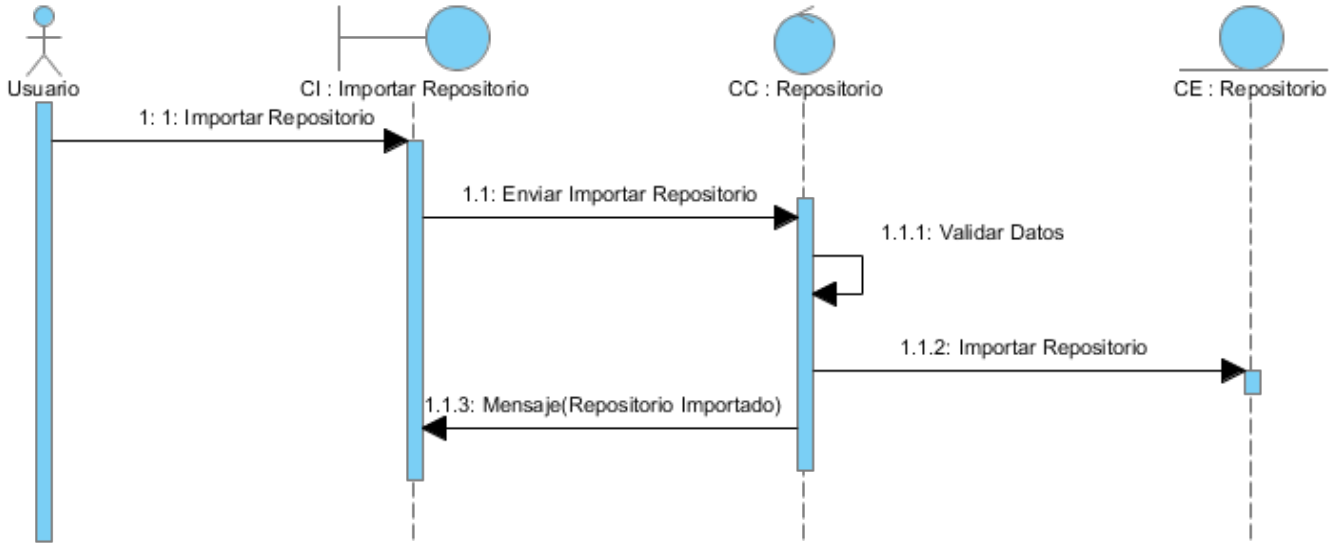
Caso de Prueba de Aceptación	
Código Caso de Prueba: HGRP-09-01	**Nombre Historia de Usuario:** Instalar paquete
Nombre de la persona que realiza la prueba: Nelio Véliz Pedraza	
Descripción de la Prueba: Se instala un paquete perteneciente a un repositorio local en el sistema.	
Condiciones de Ejecución: Debe existir al menos un repositorio local con al menos un paquete.	
Entrada / Pasos de ejecución: 1. Se selecciona un paquete en la lista de paquetes disponibles. 2. En el menú Archivo se selecciona la opción Instalar Paquete.	
Resultado Esperado: Se ejecuta el Centro de *Soft*ware.	
Evaluación de la Prueba: Satisfactoria	

Anexo 5: CPA_01: Instalar paquete

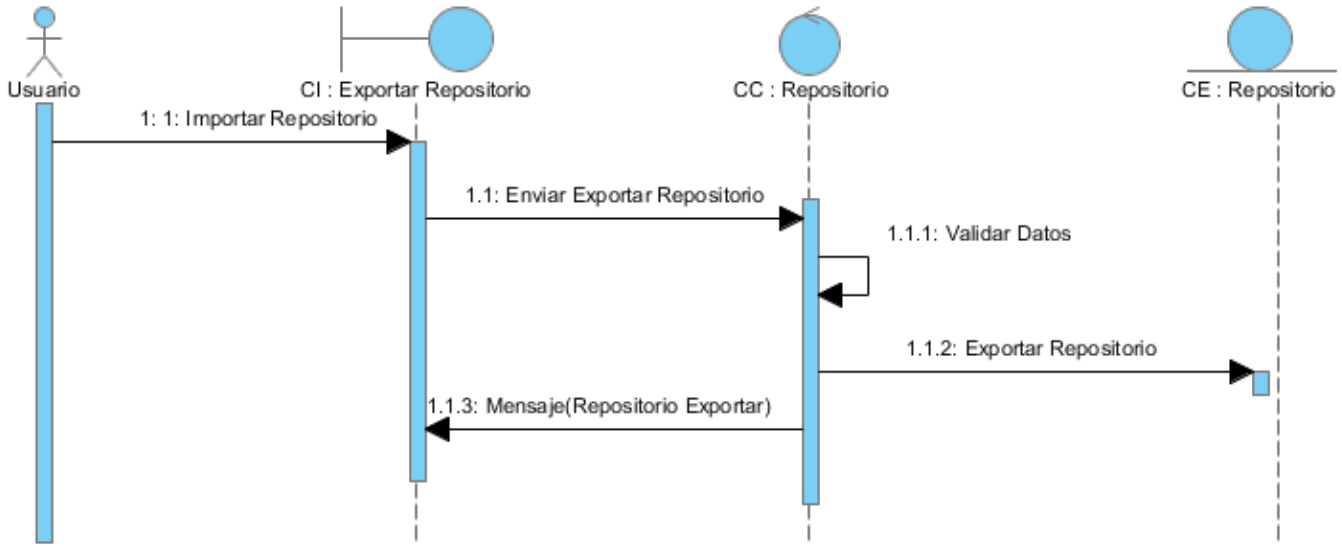
Diagramas de Secuencia:



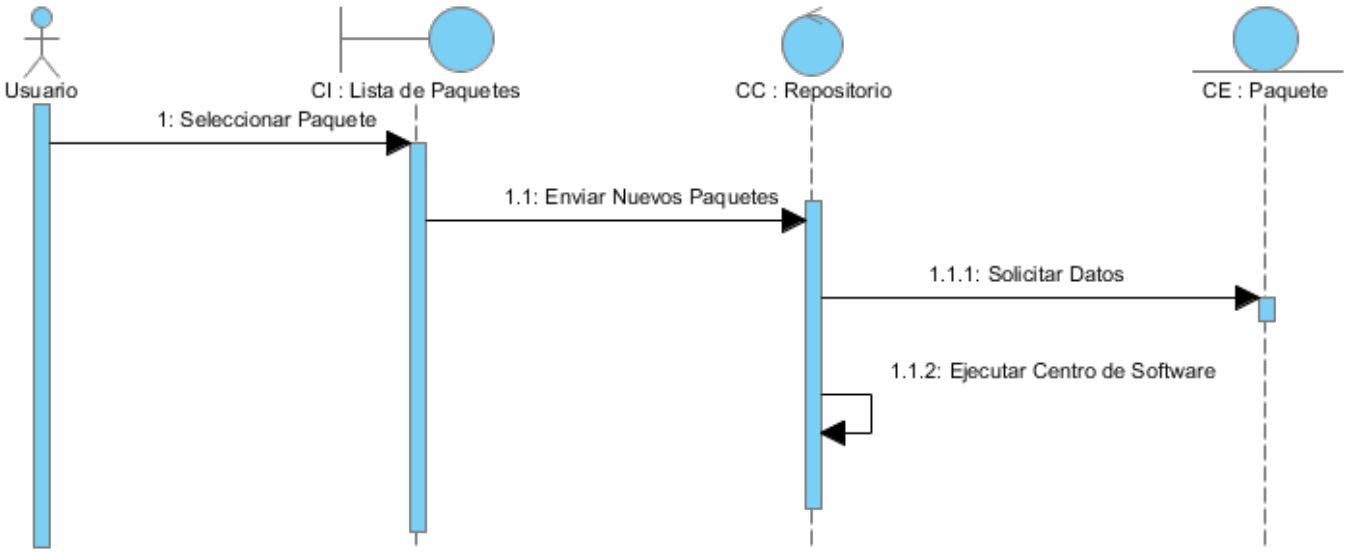
Anexo 6: DS: Configurar repositorio. Detectar repositorio.



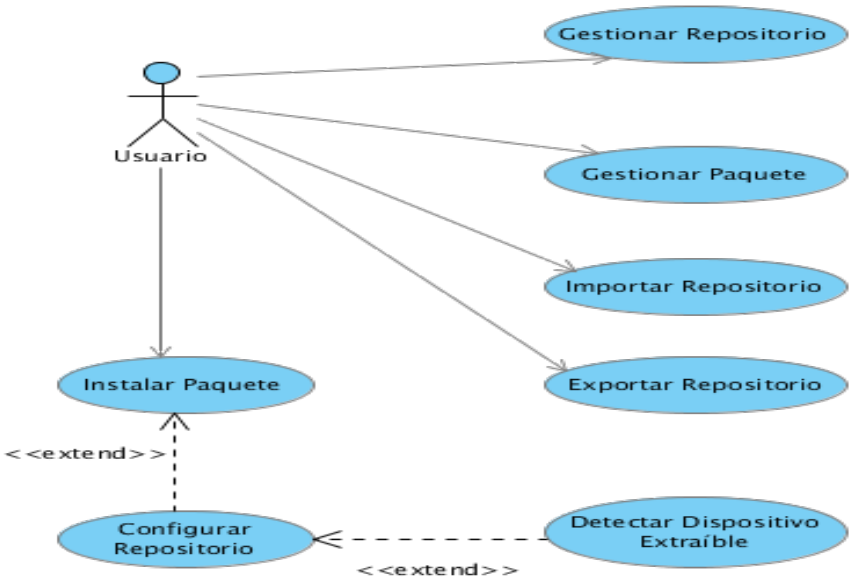
Anexo 7: DS: Importar repositorio



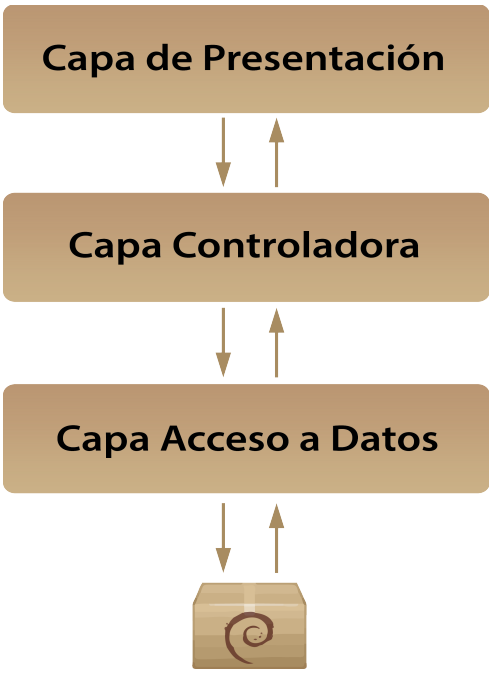
Anexo 8: DS: Exportar repositorio



Anexo 9: DS: Instalar paquete



Anexo 10: Diagrama de Casos de Uso.



Anexo 11: Arquitectura N-Capas (3 Capas)

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 utilizadas 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.

Deb: Formato de paquetes de las distribuciones basadas en *Debian*.

Demonio: Un *daemon* (nomenclatura usada en sistemas *UNIX* y UNIX-like), es un tipo especial de *proceso* informático no interactivo, que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Este tipo de programas puede ser ejecutado en forma persistente o reiniciado si se intenta matar el proceso dependiendo de la configuración del demonio y las políticas del sistema. La palabra *daemon* viene de las siglas en ingles D.A.E.MON (*Disk And Execution Monitor*).

Fuente de actualización: Ruta donde se encuentra el repositorio externo, puede ser en la web o en un dispositivo *USB*.

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.

QT: Conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario.

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

Script: Un *script* es un guión o conjunto de instrucciones. Permiten la automatización de tareas creando pequeñas utilidades. Es muy utilizado para la administración de sistemas UNIX. Son ejecutados por un intérprete de línea de comandos. Usualmente son archivos de texto.

Glosario de términos

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*.

Repositorio local: Repositorio de paquetes que se encuentra alojado en el disco duro de la computadora o en un dispositivo extraíble.

Repositorio portable: Repositorio que puede ser trasladado de un lugar a otro mediante algún dispositivo de almacenamiento masivo.

Repositorio personalizado: Repositorio que contiene solamente los paquetes seleccionados por el usuario.