Universidad de las Ciencias Informáticas.
Facultad 1

Título: Sistema de Construcción de Personalizaciones de Nova.

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

Autor: Eugenio Durán Aroche.

Tutores: Ing. Miguel Albalat Aguila.

Ing. Jorge Luis Machin Castillo.

Junio, 2012.

Ciudad de la Habana, Cuba.

PENSAMIENTO

DECLARACIÓN DE AUTORÍA.

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

Para que así conste firmamos la presente a los ___ días del mes de _____del año 2012.

Eugenio Durán Aroche

_____

Firma del Autor

Miguel Albalat Águila

_____

Firma del Tutor

Jorge Luis Machín Castillo

_____

Firma del Tutor

AGRADECIMIENTOS

A la primera y principal persona que tengo que agradecer todo lo que he logrado y posiblemente todo lo que lograré es a mi madre, sin su ayuda no he hubiese podido lograr todo lo que he logrado a lo largo de mi vida como estudiante y en otras esferas de la vida, ella es la autora de lo que soy y se lo agradezco de todo corazón. Otra persona a la que le debo un agradecimiento especial es a mi abuela que la quiero mucho. También a toda mi familia por ser tan unida y estar presente en todo momento.

Agradecer a la vicedecana de la facultad 1 Yenisleydi Cariaga Cristo, por la ayuda brindada. A los profesores del proyecto Nova, incluyendo a los tutores de este trabajo que hicieron posible su realización. Y agradecer a esos compañeros que siempre están dispuestos a aportar su ayuda, a todos ellos mi más sincero agradecimiento.

DEDICATORIA

Le dedico la presente tesis a:

Mi madre.

Mi abuela.

Mis hermanos.

A “La Universidad de las Ciencias Informáticas”.

RESUMEN

Nova es un sistema operativo de código abierto desarrollado en Cuba. Ante la tarea de migración a Software Libre, se hace necesaria la personalización de dicho sistema, debido mayormente a que los clientes tienden a temer al cambio. Nova al ser un sistema libre presenta las características de los mismos, siendo por ello robusto, estable y rápido. Además, puede ser distribuido, modificado, copiado y usado sin costo alguno. Y tiene ideales de libertad y soberanía al igual que los países socialistas.

El presente trabajo describe el desarrollo de un Sistema de Construcción de Personalizaciones de Nova (en lo adelante SCPN), que satisface la necesidad de automatizar el proceso de construcción de sistemas GNU/Linux a la medida. Contiene un estudio realizado sobre las principales herramientas de construcción de personalizaciones, así como de la metodología seleccionada para el desarrollo de la solución. Además se seleccionan y detallan las herramientas utilizadas.

Como resultado se obtiene una herramienta que permite crear personalizaciones de Nova, brindando una interfaz amigable al usuario, permitiéndole seleccionar los programas que desea incluir. El desarrollo de este trabajo mejorará el proceso de construcción de personalizaciones de Nova, facilitando así el proceso de migración a *software* de código abierto.

Palabras clave: Herramienta, personalización, GNU/Linux, Nova.

Tabla de Contenido

INTRODUCCIÓN.

Los ordenadores se han convertido en una herramienta fundamental en el desarrollo de la sociedad, contribuyen no solo a facilitar el trabajo, sino también funcionan como medio de entretenimiento e información. Según se van sofisticando los sistemas informáticos ha ido surgiendo la necesidad real de su personalización.

En la actualidad el desarrollo de la mayoría de los sistemas operativos tiene un enfoque genérico, lo que tiene como objetivo principal brindar un mayor soporte de *hardware* y se pueda utilizar en varios dispositivos que no sea precisamente una computadora tradicional, entre ellos *tablets* y celulares. Al aumentar el soporte de dichos sistemas genéricos, disminuye su explotación óptima, surgiendo como solución el desarrollo de los sistemas operativos a la medida, que son personalizaciones para un entorno determinado.

El proyecto productivo Nova que pertenece al Centro de Software Libre (CESOL), ubicado en la facultad 1, se dedica al desarrollo del sistema operativo Nova, este sistema tiene código abierto y es genérico. En el proceso de migración a software libre que se lleva a cabo en el país, Nova es uno de los sistemas operativos que se está desplegando, ya que además de ser cubano, cuenta con las cuatro libertades de los sistemas GNU/Linux.

Debido a que las personas tienden a temer al cambio, con frecuencia en el proyecto Nova se realizan personalizaciones, con el objetivo de facilitar la adaptabilidad del cliente. A pesar de ser un proyecto relativamente joven, ha alcanzado importantes resultados en la personalización de sistemas GNU/Linux, ejemplos de ello son las que se han desarrollado para: **MENPET[1], AGN[2]**, **AIN[3]**, y para competencias de programación realizadas en Cuba convocada por la **ACM-ICPC[4].**

Este engorroso proceso de personalización actualmente se desarrolla de forma manual, teniendo como principales inconvenientes la necesidad de numerosos recursos humanos y materiales y en ocasiones no se satisfacen todas las necesidades existentes.

Partiendo de la problemática anterior se presenta el siguiente **problema a científico:** ¿cómo mejorar el proceso de construcción de personalizaciones de sistemas GNU/Linux en el proyecto Nova?, para orientar la investigación se identifica como **objeto de estudio:** el proceso de construcción de personalizaciones de sistemas operativos **GNU/Linux**, y para enmarcar el alcance del mismo se cuenta con el **campo de acción:** el proceso de construcción de personalizaciones en el Proyecto

[1] Ministerio de Energía y Petróleo.
[2] Archivo General de la Nación.
[3] Agencia Internacional de Noticia.
[4] ACM *International Collegiate Programming Contest, abreviado* ACM-ICPC o simplemente ICPC.

Nova.

Para realizar la investigación se trazó el **objetivo general:** Desarrollar un sistema que facilite la construcción de personalizaciones de Nova, contando con los **objetivos específicos:**

1. Analizar la literatura existente sobre el proceso de construcción de personalizaciones de sistemas operativos.
2. Identificar y diseñar los componentes de la aplicación.
3. Implementar el sistema.
4. Realizar pruebas al sistema evaluando las funcionalidades implementadas.

Para agilizar y organizar el presente trabajo se definieron las **tareas de investigación:**

1. Revisión bibliográfica acerca del proceso de construcción de personalizaciones de sistemas operativos.
2. Selección de las herramientas pertinentes y elaboración del diseño de la aplicación.
3. Implementación del sistema construcción de personalizaciones de sistemas operativos.
4. Elaboración y aplicación de casos de prueba.

Se define como **idea a defender:** el desarrollo de un Sistema de Construcción de Personalizaciones puede mejorar el proceso de desarrollo de soluciones a la medida en el proyecto Nova.

Para el desarrollo de esta investigación, se usaron algunos de los métodos científicos existentes:

Método Teórico.

Analítico-Sintético: Para la realización de la investigación el autor realizó un estudio de un grupo de herramientas de personalización de sistemas operativos, que fueron analizados de manera independiente, estudiando minuciosamente cada uno de ellos y sintetizando sus contenidos para su mejor comprensión [1].

Métodos empíricos.

La observación: Para analizar como las principales herramientas de personalización de sistemas operativos desarrollan este proceso, obteniendo conocimientos que serían usados en la implementación [1].

Como técnica de recopilación de información se utilizó la **entrevista:**

Esta técnica se utilizó para adquirir conocimientos importantes a partir de la conversación con un especialista del proyecto Nova sobre temas de construcción de personalizaciones de sistemas operativos, que luego se llevarán a la práctica (Ver anexo #1).

El presente trabajo de diploma está compuesto por 3 capítulos:

Capítulo 1 - Fundamentación teórica: En este capítulo se desarrolla la fundamentación teórica del trabajo de diploma. Se realiza una descripción de las principales herramientas que se utilizan, así

como los lenguajes con que se implementan el sistema. Se define la metodología a seguir y se realiza la descripción de la propuesta de solución.

Capítulo 2 - Análisis y Diseño de la propuesta de solución: En este capítulo se definen los requisitos del sistema. Se realiza el Análisis y Diseño, se definen los patrones a seguir, se diseñan los casos de uso, entre otros puntos de vital importancia para el desarrollo de la investigación.

Capítulo 3 - Implementación y Pruebas: Se documenta cómo se implementó el SCPN y se le elaboran y aplican las pruebas funcionales al sistema.

Además el documento cuenta con las sesiones: Conclusiones, Recomendaciones, Referencias Bibliográficas, Bibliografía, Anexos y Glosario de Términos.

Capítulo 1: Fundamentación teórica.

Las herramientas de personalización [2] de sistemas operativos son aquellas que permiten al usuario crear su propia personalización de un determinado sistema operativo, añadiéndole los cambios o configuraciones de acuerdo con sus necesidades. En ocasiones se necesita tener optimizado el sistema por diversos motivos, ya sea por contarse con un *hardware* obsoleto o para realizar un trabajo específico donde solo se necesitan los programas indispensables.

1.1 Conceptos fundamentales.

Programa informático: Es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas dictadas por el programador en una computadora [3].

Sistema operativo: Es un programa o conjunto de programas que en un sistema informático gestiona los recursos de *hardware* y provee servicios a los programas de aplicación, o sea, proporcionan una plataforma de *software* encima de la cual otros programas puedan funcionar [4].

Repositorio de software: La definición de repositorio es fundamental, se puede definir como una colección de *softwares* ordenados, clasificados y disponibles para su uso con herramientas compatibles que lo usen para descargar, instalar y manipular dichos software. Dichos repositorios tienen datos a usar y metadatos[5] que procesar para por ejemplo realizar búsquedas sin recorrer rutas completas de forma remota [5].

1.2 Herramientas de personalización de distribuciones.

En este epígrafe, se realiza un estudio de las herramientas más usadas en las personalizaciones de la distribución GNU/Linux Ubuntu, que es la que sirve de base al sistema operativo Nova.

Es importante destacar, que la herramienta NIM, fue desarrollada en el proyecto Nova, sin embargo, se consideró importante revisar el estado del arte del tema a nivel internacional.

UCK.

Es una herramienta que permite personalizar Live CD/DVD[6] de Ubuntu, además de otras distribuciones basadas en Ubuntu a sus necesidades. Crea Live CD/DVD de arranque con idiomas predefinidos basados en el original usando asistente gráfico. Permite la creación de Live CD/DVD con

[5] Literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos, llamado recurso. El concepto de metadatos es análogo al uso de índices para localizar objetos en vez de datos.

[6] Un "LiveCD/DVD" es una distribución de Linux que funciona sin necesidad de instalarla en el ordenador. Utiliza la memoria RAM del ordenador para 'instalar' y arrancar la distribución en cuestión.

características especiales mediante secuencias de comandos. Es capaz de personalizar el sistema de archivos raíz (por ejemplo, instalar / desinstalar paquetes), modificar contenido (añadir / eliminar documentos, cambiar nombres de archivos) y al initrd (añadir módulos, cambiar secuencia de booteo) [6] (Ver anexo #2).

Reconstructor.

Reconstructor es una herramienta para crear CDs de Ubuntu personalizados. Utiliza cualquier versión (sea la *Desktop*, *Alternate* o *Server*) como base. Permite personalizar casi cualquier aspecto de la distribución: desde el *software* incluido hasta el aspecto visual (incluyendo temas, fuentes, fondos de pantalla, etc.). A continuación se listan algunas de sus funcionalidades [7] (Ver anexo #3):

Boot Screen: Permite entre otras cosas cambiar la pantalla de inicio.

Gnome: Permite cambiar algunos aspectos visuales de GNOME. Posibilita configurar el tema, la pantalla de login, el fondo de escritorio, las fuentes, los bordes, los iconos, etc.

Apt: Permite elegir los repositorios Apt que quieras incluir en tu disto.

Optimization: Optimización avanzada para usuarios con altos conocimientos de GNU/Linux.

LiveCD: Esta pestaña nos permite personalizar algunas cuestiones vinculadas a la utilización del ISO como Live CD.

ISO Master.

ISO Master es una aplicación gratuita, de código abierto y multiplicadora. La primera versión fue liberada como beta en agosto de 2006 basada en GTK GUI. Esta herramienta permite extraer el contenido de CDs y DVDs a una imagen ISO, permite crear una nueva imagen de disco con datos almacenados en nuestro ordenador o modificar los datos almacenados en un archivo ISO. Su interfaz está especializada en la búsqueda, creación y edición de imágenes ISO, poniendo a nuestra disposición un panel de navegación para nuestro sistema de archivo y otro panel con los datos almacenados (o que vamos a almacenar) en nuestra imagen ISO [8] (Ver anexo #4).

Estas tres herramientas como se mencionó anteriormente, pueden realizar una serie de configuraciones en el sistema de fichero del sistema operativo, lo cual es ventajoso. Sin embargo, tienen como desventaja que toma el sistema base desde un Live CD/DVD previamente instalado, que puede o no estar actualizado y de que muchas veces no se conoce su origen y pone en riesgo la seguridad de cierta institución. Por estas razones el investigador piensa que ninguna de estas herramientas debe seleccionarse como solución a la problemática existente.

NIM.

Es una herramienta desarrollada por el proyecto productivo Nova. La primera versión salió a principios del año 2011, con su código abierto, desarrollado en el lenguaje de programación Bash. Para el

correcto funcionamiento de este sistema se le pasa como parámetros, un repositorio de *software*, un perfil de sistema operativo, y una lista de paquetes a instalar, obteniendo como resultado una imagen ISO lista para ser instalada (Ver anexo #5).

En el presente sistema se tomó como base en particular el uso de la herramienta NIM, ya que esta brinda la posibilidad de desarrollo de distribuciones GNU/Linux que se adapten a las necesidades del usuario, además de beneficiarse de la ventaja de tomar el sistema base de un repositorio de *software*, donde los programas y paquetes se mantienen actualizados y seguros de ataques informáticos. Dicha herramienta permite al usuario introducir el repositorio con que quiere trabajar y los programas que desea instalar. La deficiencia de esta herramienta radica en el hecho de que su uso es a través de una *shell*[7] de Linux y se requiere permisos de administrador para su uso.

Se seleccionó esta herramienta, porque, entre otras ventajas, toma el sistema base de un repositorio, cosa que no hacen las anteriores. Y es más ventajoso trabajar con un repositorio de software que con un CD/DVD previamente instalado. Además, al ser desarrollada en el propio Proyecto Nova, se ajusta a las características y requerimientos necesarios para cualquier aplicación relacionada con este sistema operativo.

1.3 Metodología de desarrollo ágil.

Las metodologías ágiles forman parte del movimiento de desarrollo ágil de software, que se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto. Se le denomina ágil como la habilidad de responder de forma versátil al cambio para maximizar los beneficios [9]. Las metodologías ágiles varían en su forma de “responder al cambio”, pero en general comparten características como las siguientes:

- Usar procesos de construcción iterativos
- Entregar software funcional lo más pronto posible.
- Privilegiar el valor de la gente sobre el valor del proceso.
- Fortalecer la comunicación y la colaboración.
- Las metodologías ágiles, dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. De forma que una metodología ágil es la que tiene como principios que: Los individuos y sus interacciones son más importantes que los procesos y las herramientas.

[7] Es el intérprete de comandos que sirve de interfaz entre el usuario y el sistema operativo (en inglés "***shell***" significa "caparazón"). La shell es un archivo ejecutable que debe interpretar los comandos, transmitirlos al sistema y arrojar el resultado.

- El software que funciona es más importante que la documentación exhaustiva.
- La colaboración con el cliente en lugar de la negociación de contratos.
- La respuesta delante del cambio en lugar de seguir un plan cerrado.

Se decidió trabajar con una metodología ágil y no pesada ya que presenta las siguientes ventajas:

- Rápida respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos cortos de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Minimiza los costos frente a cambios.
- Importancia de la simplicidad, al eliminar el trabajo innecesario.
- Mejora continua de los procesos y el equipo de desarrollo.
- Evita malentendidos de requerimientos entre el cliente y el equipo.
- El equipo de desarrollo no malgasta el tiempo y dinero del cliente desarrollando soluciones innecesariamente generales y complejas que en realidad no son un requisito del cliente.
- Cada componente del producto final ha sido probado y satisface los requerimientos.

Esta metodología tiene como desventajas:

- Falta de documentación del diseño.
- Problemas derivados de la comunicación oral. Este tipo de comunicación resulta difícil de preservar cuando pasa el tiempo y está sujeta a muchas ambigüedades.
- Falta de reusabilidad. La falta de documentación hacen difícil que pueda reutilizase el código ágil [9].

Se va a usar la metodología de desarrollo OpenUp debido al poco personal que trabaja en el desarrollo del sistema, el límite de tiempo de la entrega del producto relativamente corto y que es la metodología que se sigue en el proyecto Nova. El uso de esta metodología permitirá que se concluya el desarrollo de la investigación en el tiempo estimado.

Metodología de desarrollo de software OpenUp.

Esta metodología es relativamente nueva, lanzada al mundo en el 2006 por la IBM, diseñada para el desarrollo de proyectos y que ha tomado las mejores prácticas del RUP.

OpenUP es un marco del proceso del desarrollo del software Open Source que en un cierto plazo, se espera que cubra un amplio sistema de necesidades para los proyectos de desarrollo.

OpenUP es un proceso iterativo para el desarrollo de software que es:

- Mínimo: Solo incluye el contenido del proceso fundamental.
- Completo: Puede ser manifestado como proceso entero para construir un sistema.
- Extensible: Puede ser utilizado como base para agregar o para adaptar más procesos [10].

Ventajas.

- Es apropiado para proyectos pequeños y de bajos recursos permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas [10].

Desventajas.

Al ser una metodología ágil presenta la desventaja de tener restricciones en cuanto a tamaño de los proyectos abordables y problemas derivados del fracaso de los proyectos ágiles [9].

Entre sus principios, destacan el de colaborar para alinear intereses y para compartir conocimiento, balancear las prioridades para maximizar las necesidades de los stakeholder[8] , trabaja enfocado en la Arquitectura y posee un desarrollo Iterativo [10].

Se desarrolla en 4 fases:

1. **Inicio.**

   Primera de las 4 fases en el ciclo de vida del proyecto, acerca al entendimiento del propósito y los objetivos, obteniendo suficiente información para identificar que se debe hacer en el proyecto. El objetivo de ésta fase es capturar las necesidades de los stakeholder.

2. **Elaboración.**

   Es el segundo de las 4 fases del ciclo de vida del OpenUP donde se trata los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base la de elaboración de la arquitectura del sistema.

[8] En el contexto de Responsabilidad Social, el término fue utilizado por primera vez por R. E. Freeman en su obra: “Strategic Management: A Stakeholder Approach”, (Pitman, 1984) para referirse a quienes pueden afectar o son afectados por las actividades de una empresa.

3. **Construcción.**

   Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema.

4. **Transición.**

   Es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y perforador del último entregarle de la fase de construcción [10].

En la Figura 1 se muestran las fases que define OpenUp por las cuales debe transitar el desarrollo de un producto software.

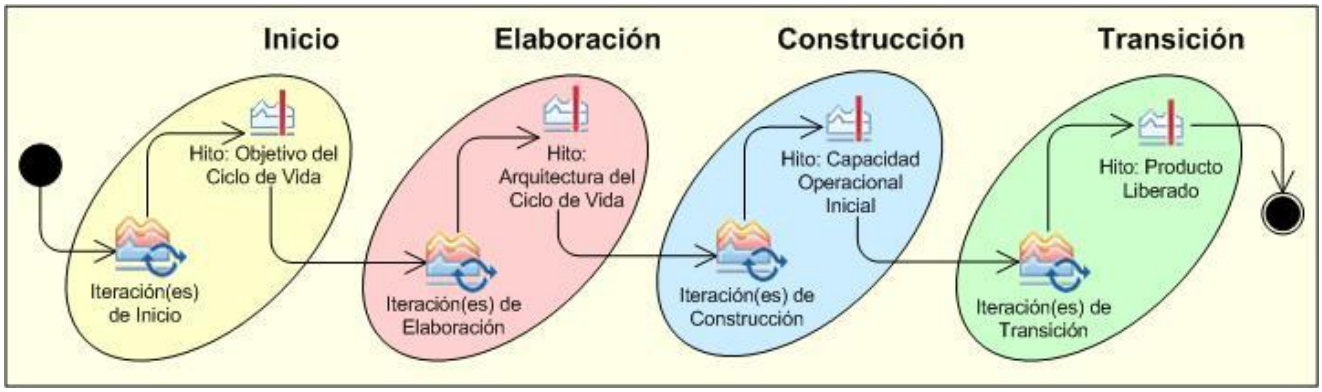


Fig. 1: Fases de OpenUp.

El OpenUp es un proceso modelo y extensible, dirigido a gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental apropiado para proyectos pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

1.4 Herramientas de desarrollo.

IDE.

Geany.

Geany es un pequeño y ligero editor de textos para programadores y desarrolladores, disponible tanto para Windows como Linux. Dispone de las funciones básicas de un editor, siendo necesario para su uso tener instalados las librerías gtk2 *runtime*.

Tiene soporte para muchos lenguajes de programación distintos, como C, C++, C#, Java, JavaScript, PHP, HTML, CSS, Python, Perl, Ruby, Fortran, Pascal y Haskell.

Algunas de las características más destacadas de Geany son: autocompletado, soporte multidocumento, soporte de proyectos, coloreado de sintaxis y emulador de terminal incrustado [11].

Las características de este IDE que permitieron su uso en la implementación del sistema fueron: es ligero lo que conlleva a un trabajo relativamente rápido, tiene soporte para varios lenguajes, esto permite que se esté programando en varios lenguajes en un mismo IDE, además tiene autocompletado de código, lo que facilita la programación. La versión del Geany seleccionada para desarrollar es la 0.18.

Herramienta de modelado.

Visual Paradigm.

Visual Paradigm es una herramienta multiplataforma que cuenta además, con versiones gratuitas y provee fácil integración con el resto de las herramientas de desarrollo. Ayuda a una rápida construcción de aplicaciones calidad y con el menor costo posible. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación.

Visual Paradigma es un producto que facilita la organización para el diseño visual de diagramas.

Visual Paradigma ofrece:

- ✓ Entorno de creación de diagramas para UML 2.0.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- ✓ Disponibilidad de múltiples versiones, para cada necesidad.
- ✓ Disponibilidad de integrarse en los principales Idea.
- ✓ Disponibilidad en múltiples plataformas [12].

Esta herramienta **CASE** es seleccionada para el modelado del SCPN, y esto se debe en gran medida a que posee un paquete para el trabajo con metodologías ágiles, además de utilizar **UML**. La versión que se eligió es la 6.4.

Lenguajes de desarrollo.

Perl.

Perl es un lenguaje interpretado optimizado para la lectura y extracción de información de archivos de texto, generando reportes basados en la información proporcionada por ellos. Es también un lenguaje bastante utilizado para muchos sistemas manipuladores de tareas como lenguaje de contenido dinámico. Este lenguaje tiene una orientación más práctica (facilidad de uso y eficiencia) que estética (elegancia, minimalista) [13].

Fue creado en octubre de 1987 por Larry Wall. El propio autor señala que combina algunas de las mejores capacidades de C, sed, awk y sh, por lo que programadores familiarizados con estos lenguajes tendrán gran facilidad para trabajar con él sobre todo en administración de sistemas.

Perl tiene posee una gama muy amplia de usos, en una gran variedad de arquitecturas, bajo varios sistemas operativos. Es con seguridad uno de los lenguajes más utilizados para tareas pequeñas y de tamaño mediano, dada la rapidez de aprendizaje de que ostenta, su capacidad de modularización lo hace también un candidato factible para tareas de mayor envergadura.

Provee herramientas muy sencillas de uso para la generación de páginas dinámicas vía procesamiento de información de entrada/salida a través de la Web. Este hecho masificó el uso de Perl en la creación de scripts[9] para la web, así como un sinnúmero de otras aplicaciones. La versión seleccionada en la implementación del sistema es Perl 5.10.1.

Python.

Python es un lenguaje de programación de alto nivel cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca a un código legible. Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y, en menor medida programación funcional [14].

Es administrado por la *Python Software Foundation*. Posee una licencia de código abierto, denominada *Python Software Foundation License*, que es compatible con la Licencia pública general de GNU a partir de la versión 2.1.1, e incompatible en ciertas versiones anteriores. La versión seleccionada en la implementación del sistema es Python 2.6.

Bash.

Es un programa informático cuya función consiste en interpretar órdenes. Está basado en la shell de Unix y es compatible con POSIX. Fue escrito para el proyecto GNU y es el intérprete de comandos por defecto en la mayoría de las distribuciones de Linux [15]. La versión utilizada en la implementación del sistema es Bash 4.1.

Para implementar el SCPN se decidió usar estos lenguajes de programación ya que posibilitará hacer un sistema robusto y con una interfaz amigable al usuario. Además de brindar flexibilidad y poderse seguir desarrollando en un futuro.

En este caso en el proyecto, se definió como requerimiento que el SCPN se integrara a un modelo de interfaz que debe ser desarrollado en el lenguaje Python esencialmente porque forma parte de una plataforma de construcción de sistemas operativos. Sin embargo existen algunas herramientas que se utilizan para su desarrollo que tienen como lenguaje Bash o Perl y para facilitar la comunicación entre ellas, se utilizan también estos lenguajes.

---

[9] Un guión o script es un fichero de texto que contiene una serie de instrucciones que se pueden ejecutar en la línea de órdenes, y que se ejecutarán seguidas. El único requisito es que ese fichero de texto tenga permiso de ejecución para la persona que intenta ejecutarlo.

1.5 Propuesta del sistema.

Como se analizó anteriormente, varios son los esfuerzos que se han realizados en los Sistemas GNU/Linux relacionado con la personalización de sistemas operativos, no obstante su enfoque es general y está destinado a usuarios con conocimientos avanzado de GNU/Linux y no a un tipo de usuario en particular.

Zentyal (Ver Anexo #6) es un servidor de red unificada de código abierto. Puede actuar gestionando la infraestructura de red, como puerta de enlace a Internet (Gateway), gestionando las amenazas de seguridad (UTM), como servidor de oficina, como servidor de comunicaciones unificadas o una combinación de estas. Además, Zentyal incluye un marco de desarrollo (*framework*) para facilitar el desarrollo de nuevos servicios basados en Unix [16]. Para ver los módulos con que dispone ver Anexo #7.

Entre sus características se encuentran las siguientes:

- ✓ Gestión de redes.
- ✓ Infraestructura de red.
- ✓ *Webmail*.
- ✓ Servidor web.
- ✓ Autoridad de Certificación.
- ✓ Trabajo en grupo.
- ✓ Informes y monitorización.
- ✓ Actualizaciones de software.

La solución que aquí se propone consiste en crear un sistema de construcción de personalizaciones como un servicio más para dicha plataforma. Para su uso el usuario se conecta vía Web a Zentyal y trabaja con el módulo SCPN. Entre las principales funcionalidades de la presente propuesta está la de seleccionar el perfil y arquitectura disponible. Además debe de gestionar repositorios, aquí el usuario puede seleccionar o añadir los repositorios de software con los que desea trabajar, y en dependencia de los que seleccione se mostrarán los paquetes o programas disponibles para añadir en su CD/DVD de Nova personalizado, también permite añadir paquetes locales a la instalación.

Con esta propuesta se proveerá de una herramienta que permita al usuario sin tener un conocimiento avanzado de Linux, construir de manera ágil y fácil un Nova personalizado que se adapte a sus necesidades, y que solamente contenga las aplicaciones que sean de su utilidad, optimizando así el SO para satisfacer una mayor variedad de necesidades.

Este sistema será desarrollado inicialmente para construir personalizaciones de la distribución cubana GNU/Linux Nova, pudiéndose extender luego a otras distribuciones, lo que aumentaría el nivel de uso de Zentyal y por consiguiente Nova en los usuarios.

En este capítulo se realizó un estudio de las una serie de herramientas de personalización de distribuciones GNU/Linux, se evidenció como pueden resolver muchos problemas a la hora de construir sistemas a la medida. Se seleccionó una metodología ágil para la investigación. Se utilizarán como lenguajes de programación: Python, Perl, Bash debido a que brindan robustez y flexibilidad al sistema. Además se seleccionó como IDE al Geany por su ligereza y como herramienta de modelado el Visual Paradigm debido a que facilita el diseño visual de diagramas. Y se realizó una propuesta de solución que consiste en crear un sistema de construcción de personalizaciones como un moduló más a la plataforma Zentyal.

Capítulo 2: Análisis y Diseño de la propuesta de solución.

A continuación se desarrolla el análisis y diseño del software que se desarrollará. Se identificarán los patrones a seguir, entre otros puntos de vital importancia.

2.1 Especificación de los requisitos de Software.

La Especificación de Requisitos Software (ERS**)** es una de las tareas más importantes en el ciclo de vida del desarrollo de *software*, puesto que en ella se determinan los "planos" de la nueva aplicación. Se hace una descripción completa del comportamiento del sistema que se va a desarrollar. Y además de los casos de uso, la ERS también contiene requisitos no funcionales (o complementarios) [17].

Como resultado de esta fase se debe producir un documento de especificación de requisitos en el que se describa lo que el futuro sistema debe hacer. Al SCPN como a todo *software* también se le consideró importante realizar el documento de ERS, donde se plasma cada requisito y su comportamiento.

Requisitos Funcionales:

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. En la investigación fueron de importancia ya que facilitó el trabajo a la hora de implementar el sistema, y sirvió como guía, definiendo que se debía hacer en cada momento [17].

El sistema debe permitir:

RF1: Adicionar repositorio.

RF2: Modificar repositorio.

RF3: Eliminar repositorio.

RF4: Seleccionar perfile y arquitectura.

RF5: Seleccionar paquetes disponibles.

RF6: Construir imagen ISO.

RF7: Adicionar paquetes locales a la instalación.

RF8: Construir discos virtuales.

RF9: Enviar correo de notificación.

RF10: Hacer configuraciones.

Requisitos no Funcionales:

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Los requisitos no funcionales forman una parte significativa de la especificación [17].

Usabilidad.

RNF 1: El sistema debe presentar una interfaz amigable.

RNF 3: El sistema debe contar con un diseño gráfico a fin con el usuario.

Apariencia o interfaz externa.

RNF 4: Brindará una navegación sencilla.

Rendimiento.

RNF 5: Tiempos de respuestas relativamente rápidos.

Soporte.

RNF 6: Las pruebas al sistema deben identificar errores que puedan convertirse en defectos potenciales.

Hardware.

RNF 7: Para el servidor donde será montado el sistema se requiere: PENTIUM IV, con 2 GB de RAM, o con prestaciones mayores a estas.

RNF 8: Para el almacenamiento debe tener al menos 10Gb de capacidad de disco duro disponible.

Software.

RNF 9: El sistema debe ser instalado en el SO Nova.

RNF 10: Navegador compatible: Mozilla Firefox.

2.2 Análisis y diseño.

El Análisis y Diseño de un producto de software tiene como objetivos principales, transformar los requisitos de software en un diseño del sistema. Identificar una arquitectura sólida y adaptar el diseño para que se ajuste al entorno de implementación, con un diseño pensado para el rendimiento. La disciplina de Análisis y Diseño transformó los requisitos funcionales del SCPN en un diseño del sistema, dando una visión de cómo se debe implementar el sistema y abriendo paso a la programación de este.

Análisis.

Luego de identificados todos los requisitos que debe cumplir el software que se va a desarrollar, es importante realizar el análisis de ellos, con el objetivo de tener una mejor comprensión antes de entrar al diseño de dicho software, garantizando así una arquitectura robusta, eficaz, eficiente y capaz de sobrevivir a cambios [18].

Diseño.

El diseño tiene el propósito de formular los modelos que se centran en los requisitos no funcionales y en el dominio de la solución, además prepara para la implementación y prueba del sistema. Pretende crear un plano del modelo de implementación, por lo que el grueso del esfuerzo está en las últimas iteraciones de elaboración y las primeras de construcción [18].

Patrones de casos de uso.

Los patrones de casos de uso son comportamientos que deben existir en el sistema, describen el uso del mismo y cómo este interactúa con los usuarios. Estos patrones son utilizados generalmente como plantillas que detallan como debería ser estructurados y organizados los casos de uso. Son patrones que capturan mejores prácticas para modelar casos de uso [19].

Los patrones que se utilizaron para el desarrollo del sistema son:

Extensión concreta o Inclusión.

Consiste en dos casos de uso y una relación extendida entre ellos. Puede ser instalado en sí mismo, así como extendido en el caso de uso base. El referente puede ser concreto o abstracto. Este patrón se aplica cuando un flujo puede extender el flujo de otro caso de uso así como ser realizado en sí mismo [19].

Se incluye una relación del caso de uso base al caso de uso de inclusión. El último puede ser instalado en sí mismo. El caso de uso base puede ser concreto o abstracto.

CRUD.

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.

Completo: Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y estos a su vez son cortos y simples.

Parcial: Este patrón alternativo modela una de las vías de los casos de uso como un caso de uso separado. Es preferiblemente utilizado cuando una de las alternativas de los casos de uso es más significativa, larga o más compleja que las otras [19].

Múltiples actores.

A un Caso de Uso ingresan más de dos actores y estos tienen un rol común.

Roles diferentes: Captura la concordancia entre actores manteniendo roles separados. Consiste de un

caso de uso y por lo menos dos actores. Es utilizado cuando dos actores juegan diferentes roles en un caso de uso, o sea, interactúan de forma diferente con el caso de uso.

Roles común: Puede suceder que los dos actores jueguen el mismo rol sobre el CU. Este rol es representado por otro actor, heredado por los actores que comparten este rol. Es aplicable cuando, desde el punto de vista del caso de uso, solo exista una entidad externa interactuando con cada una de las instancias del caso de uso [19].

Actores del sistema.

Id	Descripción
Usuario	El usuario es cualquier persona que interactúe con el sistema.
Administrador	Es el encargado de darle mantenimiento al sistema, además de hacer las configuraciones necesarias.

Tabla 1: Actores del sistema.

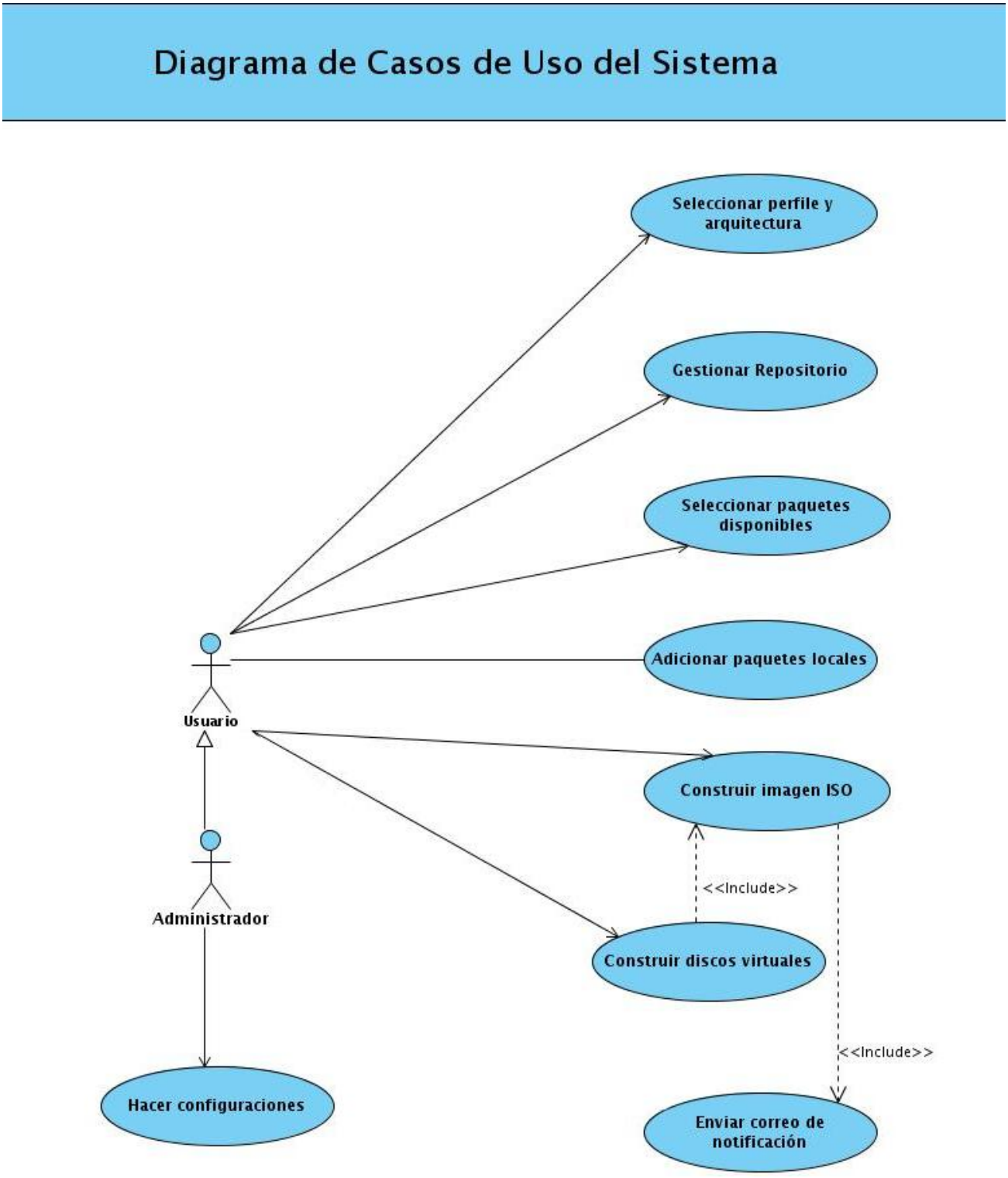


Fig. 2: Diagrama de Caso de Uso del Sistema.

2.3 Descripción textual de los casos de uso.

En las siguientes tablas serán descritos los casos de uso del sistema que fueron modelados en el diagrama anterior. En cada descripción se especifica el propósito general de cada uno de ellos, el resumen de su funcionamiento, así como las condiciones que deben existir para que estos se ejecuten.

Caso de uso	Seleccionar Perfil y Arquitectura.	
Objetivo	Permite al usuario seleccionar el perfil y arquitectura de que desee de las que están disponibles.	
Actores	Usuario.	
Resumen	El caso de uso inicia cuando el usuario selecciona la opción de Perfil y Arquitectura.	
Complejidad	Alta.	
Prioridad	Crítico.	
Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.	
Postcondiciones	Visualización de los perfiles y arquitecturas disponibles.	
Flujo Normal de Eventos.		
Acción del actor.		**Acción del sistema.**
1. Selecciona la opción Perfil y Arquitectura.		2. Muestra un formulario con los perfiles y arquitecturas disponibles.
3. Selecciona el perfil y arquitectura que quiere para su personalización.		4. Notifica la acción realizada.

Tabla 2: Descripción textual CU Seleccionar Perfil y Arquitectura.

Caso de uso	Gestionar repositorio.
Objetivo	Permite al usuario gestionar (insertar, modificar, eliminar) los repositorios disponibles.
Actores	Usuario.
Resumen	El caso de uso se inicia cuando el Usuario selecciona la opción de Gestionar Repositorio, luego selecciona el tipo de gestión, introduce los datos necesarios y el sistema realiza la acción seleccionada.
Complejidad	Alta.
Prioridad	Crítico.

Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.
Postcondiciones	Nuevo repositorio registrado en el sistema, modificado o eliminado.

Flujo Normal de Eventos.	
Acción del actor.	**Acción del sistema.**
1. Selecciona la opción Gestionar Repositorio.	2. El sistema le muestra un formulario con una lista de repositorios disponibles.
3. Selecciona una opción: a. Si selecciona la opción añadir repositorio, véase sección Insertar repositorio. b. Si selecciona modificar, véase sección Modificar repositorio. c. Si selecciona eliminar, véase sección Eliminar un repositorio.	
Sección "Insertar Repositorio".	
Acción del actor.	**Acción del sistema.**
	1. Muestra el formulario para insertar un nuevo repositorio.
2. Introduce los datos del nuevo repositorio.	3. Verifica que los datos del repositorio son correctos. 4. Si hay errores en los datos véase flujo alterno 4.1. 5. Si los datos son correctos el sistema inserta un nuevo repositorio.
Flujo Alterno 4.1	
Acción del actor.	**Acción del sistema.**
	4.1 Si hay errores en los datos del nuevo repositorio, el sistema muestra un mensaje de error y retorna el paso 1.
Sección "Modificar Repositorio".	
Acción del actor.	**Acción del sistema.**
	1. Muestra un listado de todos los repositorios.
2. Selecciona el repositorio a modificar.	3. Muestra los datos actuales del repositorio seleccionado que pueden ser modificados.
4. Modifica algunos o todos los datos del repositorio seleccionado.	5. Verifica que los datos introducidos son correctos.
	6. Si hay errores en los datos véase flujo alterno 6.1.
	7. Si los datos fueron introducidos

	correctamente, el sistema modifica los datos del repositorio seleccionado.
Flujo Alterno 6.1	
Acción del actor.	**Acción del sistema.**
	6.1. Muestra un mensaje de error y retorna al paso 3.
Sección "Eliminar Repositorio".	
Acción del actor.	**Acción del sistema.**
	1. Muestra un listado de todos los repositorios.
2. Selecciona el repositorio a eliminar.	3. Elimina el repositorio seleccionado.

Tabla 3: Descripción textual CU Gestionar Repositorio.

Caso de uso	Seleccionar Paquetes Disponibles.
Objetivo	Permite al usuario seleccionar una lista de paquetes disponibles para ser instalados.
Actores	Usuario.
Resumen	El caso de uso inicia cuando el usuario selecciona la opción de Visualizar paquetes.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.
Postcondiciones	Visualización de una lista de paquetes.
Flujo Normal de Eventos.	
Acción del actor.	**Acción del sistema.**
1. Selecciona la opción Visualizar Paquetes.	2. Muestra un formulario con una lista de paquetes disponibles.
3. Selecciona de la lista de paquetes los que se desean instalar.	4. Adiciona a las lista de paquetes seleccionados el nuevo paquete.

Tabla 4: Descripción textual CU Seleccionar paquetes disponibles.

Caso de uso	Adicionar paquetes locales.
Objetivo	Adicionar un paquete local a la instalación.
Actores	Usuario.

Resumen	El caso de uso inicia cuando el usuario selecciona la opción de Adicionar paquete local.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.
Postcondiciones	El paquete es subido al servidor donde está montado el sistema.
Flujo Normal de Eventos.	
Acción del actor.	**Acción del sistema.**
1. Selecciona la opción Subir paquete local.	2. Muestra un formulario desde donde puede introducir la dirección local de su paquete.
3. Seleccionar la dirección del paquete a adicionar.	4. Verifica que el paquete es válido.
	5. Si el paquete no es válido véase flujo alterno 5.1.
	6. Si el paquete es válido es subido por el sistema.

Flujo Alterno 5.1	
Acción del actor.	**Acción del sistema.**
	5.1 Muestra un mensaje de error y retorna al paso 2.

Tabla 5: Descripción textual CU Adicionar paquetes locales.

Caso de uso	Construir Imagen ISO.
Objetivo	Construir Imagen ISO personalizado con los paquetes seleccionado por el usuario.
Actores	Usuario.
Resumen	El caso de uso inicia cuando el usuario selecciona la opción de Construir imagen ISO.
Complejidad	Alta.
Prioridad	Crítico.
Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.
Postcondiciones	Se construye la imagen.
Flujo Normal de Eventos.	

Acción del actor.	Acción del sistema.
1. Selecciona la opción Construir imagen ISO.	2. Empieza la construcción de la imagen.
	3. Si hay errores véase flujo alterno 3.1.
	4. Se construye el ISO satisfactoriamente.

Flujo Alterno 5.1	
Acción del actor.	**Acción del sistema.**
	3.1 Muestra un mensaje de error.

Tabla 6: Descripción textual CU Construir imagen ISO.

Caso de uso	Construir Disco Virtuales.
Objetivo	Construir discos virtuales con Nova instalado.
Actores	Usuario.
Resumen	El caso de uso inicia cuando el usuario selecciona la opción de Construir discos virtuales.
Complejidad	Alta.
Prioridad	Media.
Precondiciones	El usuario debe estar autenticado en la plataforma Zentyal.
Postcondiciones	Se construye la imagen de disco duro virtual.
Flujo Normal de Eventos.	
Acción del actor.	**Acción del sistema.**
1. Selecciona la opción Construir disco virtual.	2. Empieza la construcción del disco.
	3. Si hay errores véase flujo alterno 3.1.
	4. Se construye el disco satisfactoriamente.

Flujo Alterno 5.1	
Acción del actor.	**Acción del sistema.**

	3.1 Muestra un mensaje de error.

Tabla 7: Descripción textual CU Construir discos virtuales.

Diagrama de clases del análisis.

Uno de los principales artefactos del análisis es el Diagrama de clases del análisis. En este se representan las clases de análisis (clase interfaz, clase controladora y clase entidad) y la relación que existe entre estos [20].

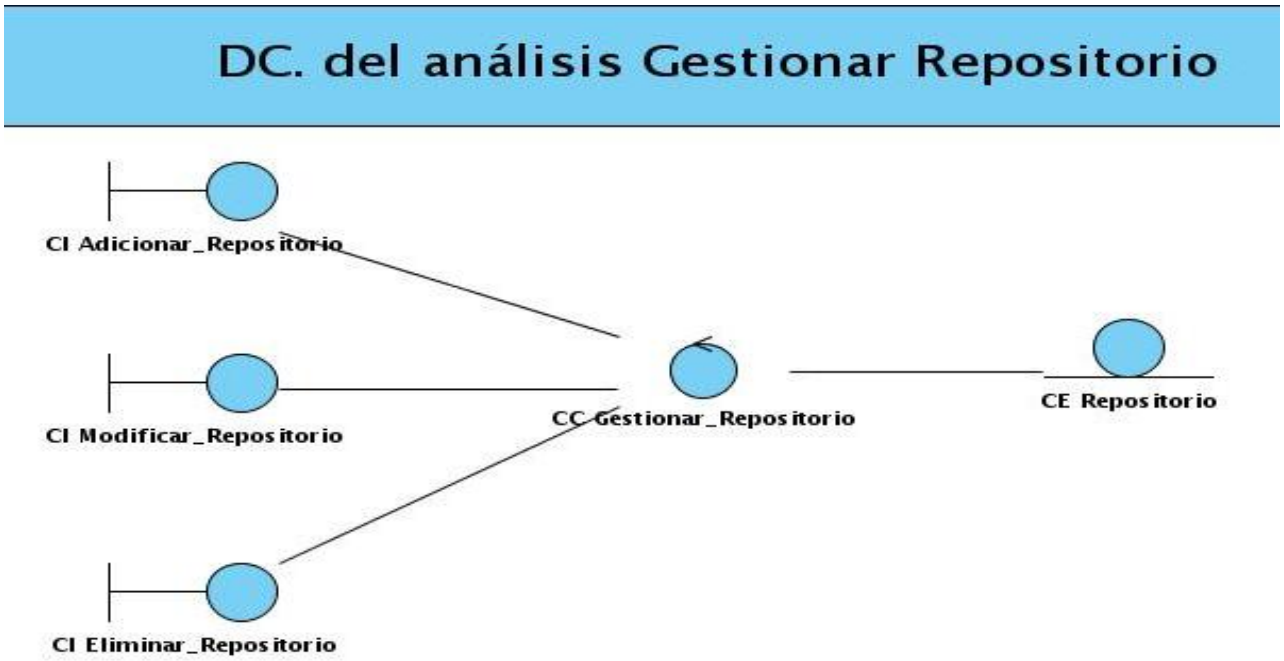


Fig. 3: Diagrama de Clases del Análisis Gestionar Repositorio.

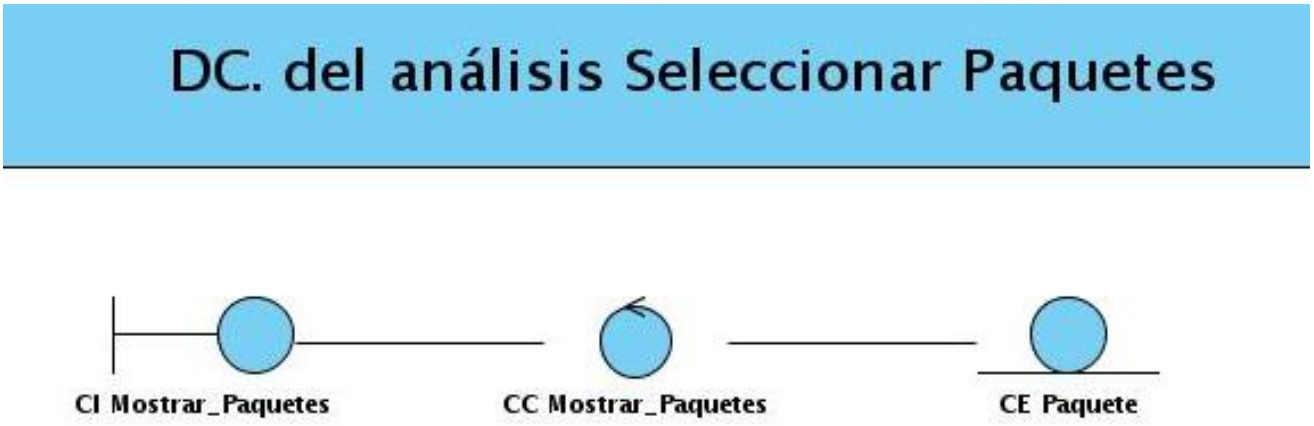


Fig. 4: Diagrama de Clases del Análisis Seleccionar Paquetes.

Diagrama de secuencia.

Al igual que los Diagramas de Colaboración los diagramas de secuencia se utilizan para ilustrar la realización de un CU. Son particularmente importantes para los diseñadores pues aclaran los roles jugados por los objetos en un flujo, lo cual le proporciona un gran valor para la determinación de las responsabilidades de las clases [21].

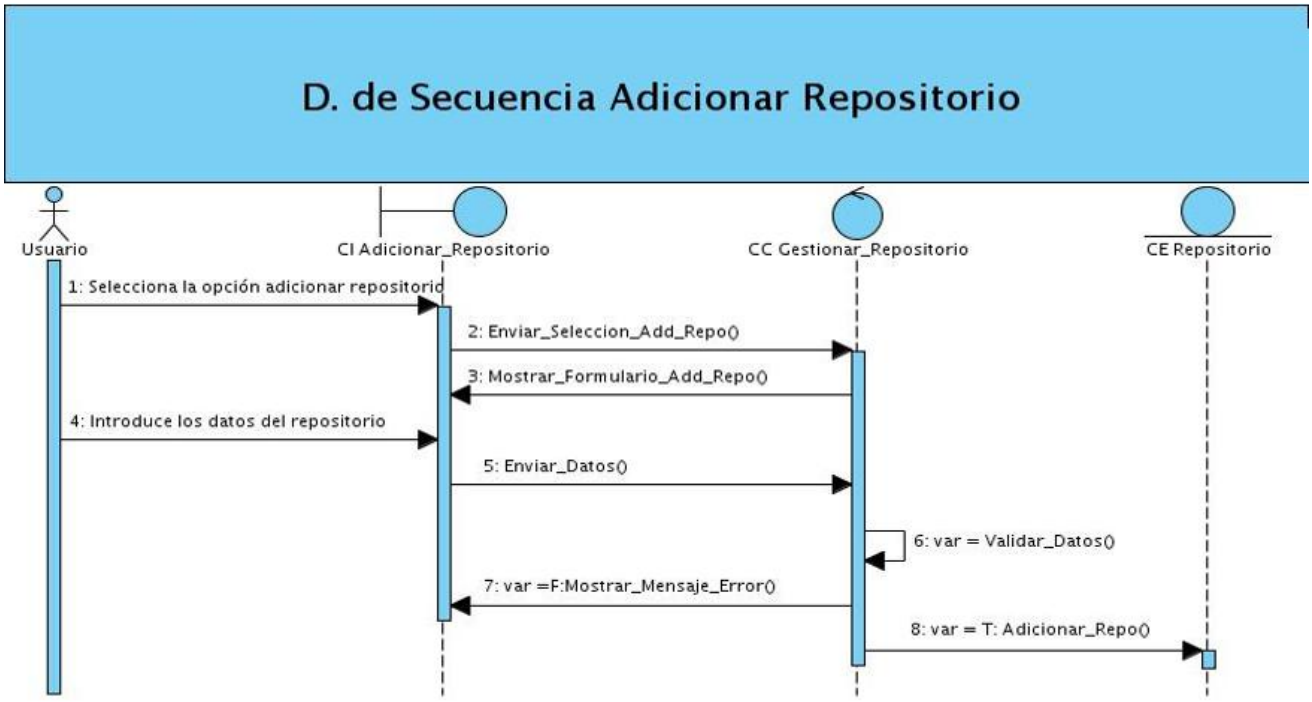


Fig. 5: Diagrama de Secuencia Adicionar Repositorio.

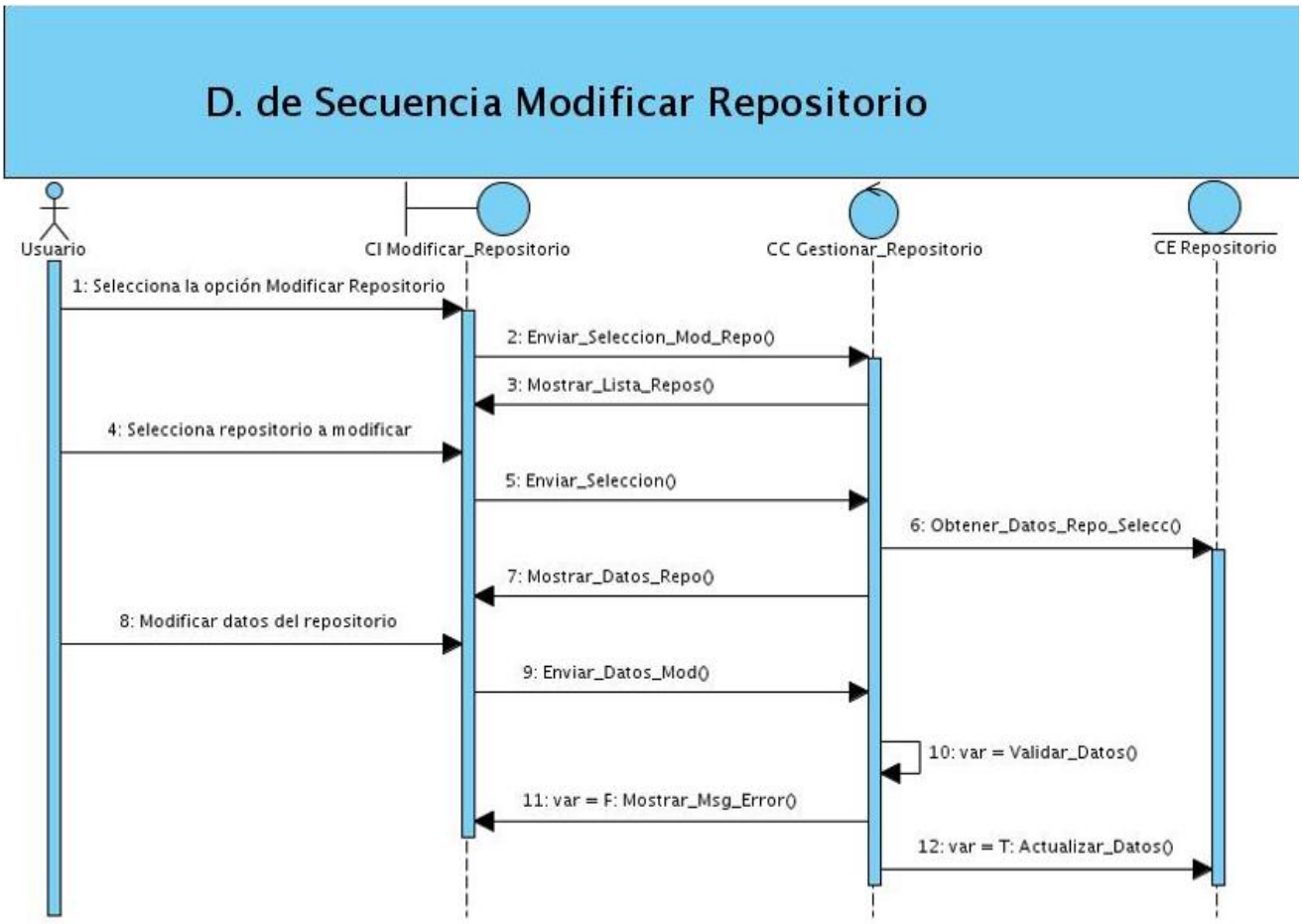


Fig. 6: Diagrama de Secuencia Modificar Repositorio.

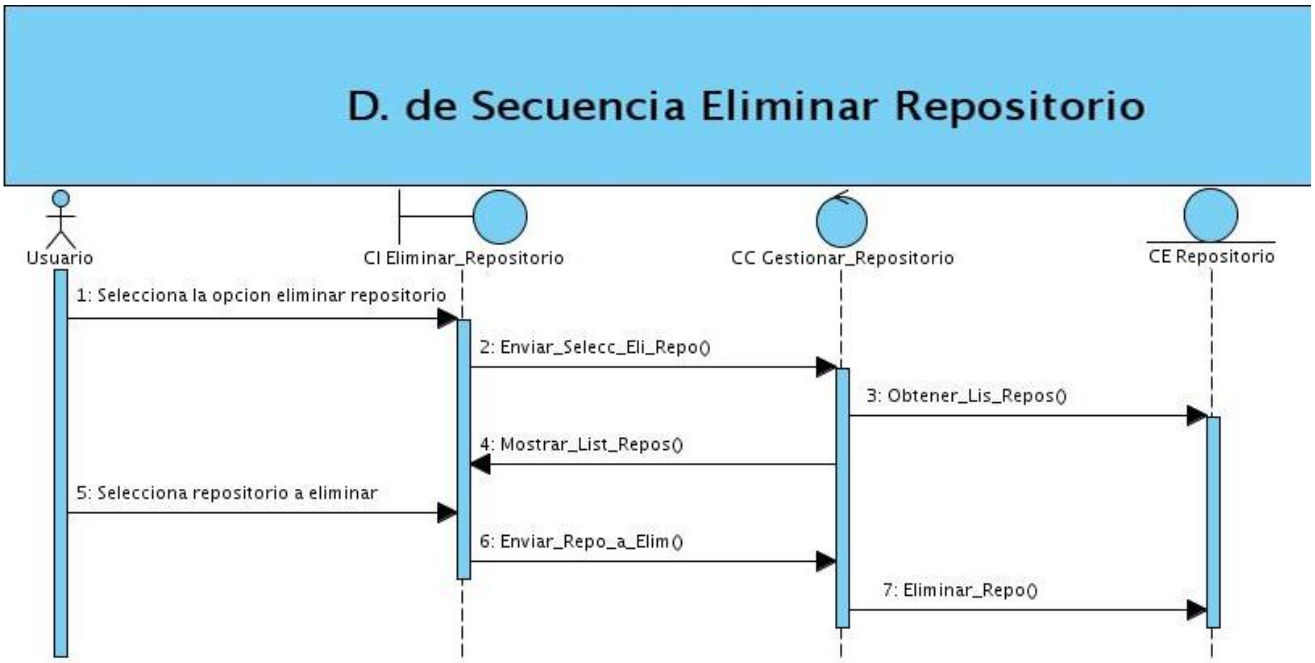


Fig. 7: Diagrama de Secuencia Eliminar Repositorio.

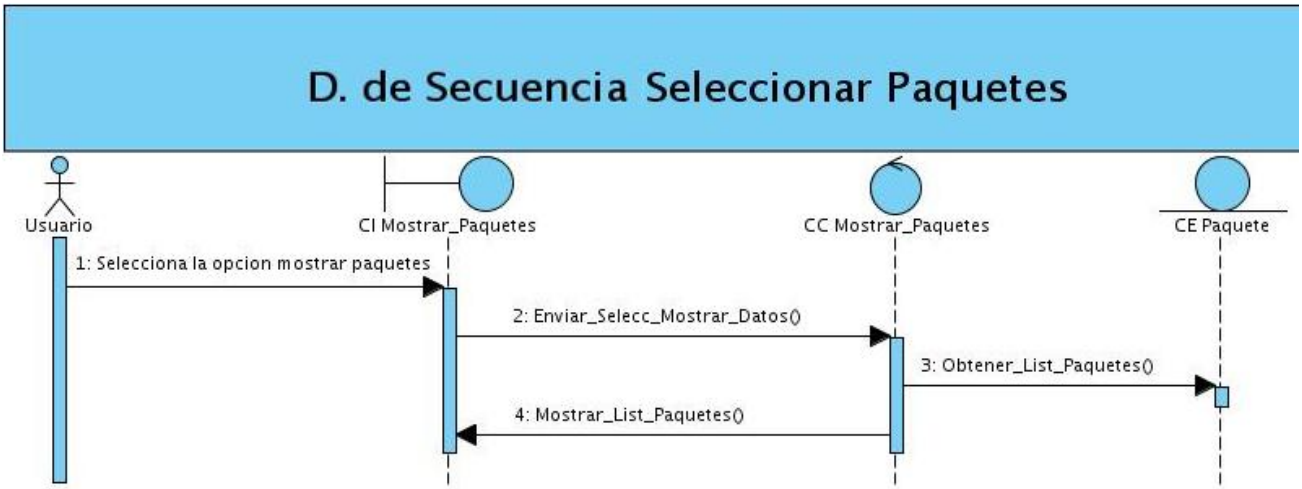


Fig. 8: Diagrama de Secuencia Seleccionar Paquetes.

2.4 Patrón de diseño.

"Los patrones de diseño son el esqueleto de las soluciones a problemas comunes en el desarrollo de software." En otras palabras, brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares [22].

Los patrones **GRASP** describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable [23].

Los patrones **GRASP** que se utilizaron son:

EXPERTO: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene diseñada [23].

CREADOR: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente cuando:

- B contiene a A.
- B es una agregación (o composición) de A.
- B almacena a A.
- B tiene los datos de inicialización de A (datos que requiere su constructor).
- B usa a A [23].

CONTROLADOR: Normalmente un controlador delega en otros objetos el trabajo que se necesita hacer; coordina o controla la actividad. No realiza mucho trabajo por sí mismo [23].

2.5 Modelo Vista Controlador.

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software, diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas complejos, este trata el Modelo, las Vistas y los Controladores como entidades separadas.

El **modelo** contiene las funciones llamadas "lógica de negocio", o sea datos y reglas de negocio. Lleva un registro de las vistas y controladores del sistema. Es el responsable de acceder a la capa de almacenamiento de datos. Tener el modelo bien delimitado permite la existencia de varias aplicaciones que compartan el mismo modelo.

La **vista** recibe los datos del modelo y los muestra al usuario. Tiene un registro de su controlador asociado. Pueden dar el servicio de "Actualización", para que sea invocado por el controlador o por el modelo.

El **controlador** recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.). Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas [24].

Principales ventajas del Modelo Vista Controlador.

- Fácil y flexible estructuración del código

- Soporte de múltiples vistas, dado que la vista se halla separada del modelo y no hay dependencia directa entre estas.
- Adaptación al cambio, los requerimientos de interfaz de usuario tienden a cambiar con frecuencia, pero esto no sería ningún problema ya que el modelo no depende de las vistas, y agregar nuevas funciones generalmente no afecta al modelo [25].

2.6 Diagrama de clases.

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro [26].

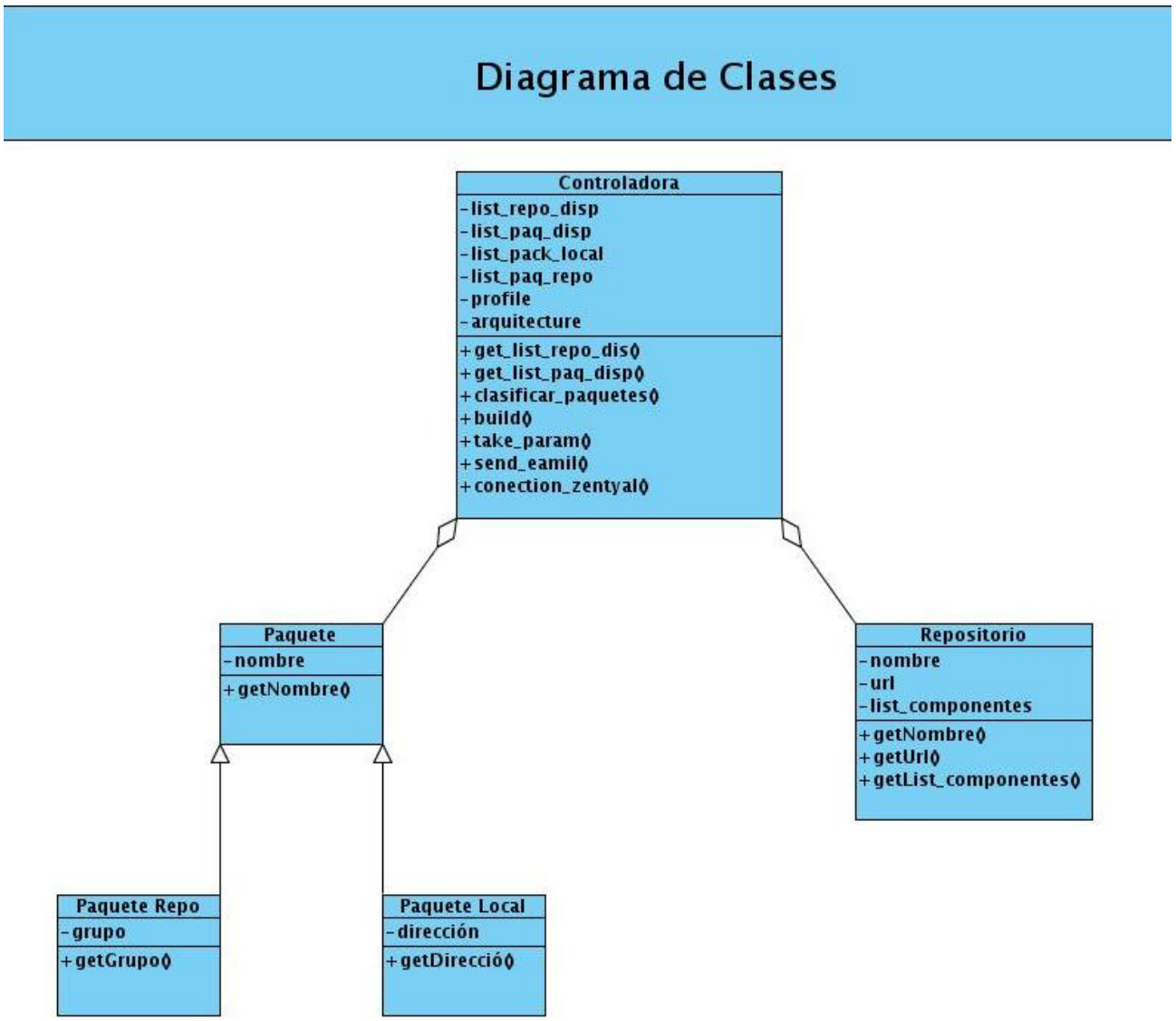


Fig. 9 Diagrama de Clases.

2.7 Diagrama de Componentes.

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los Diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema [27].

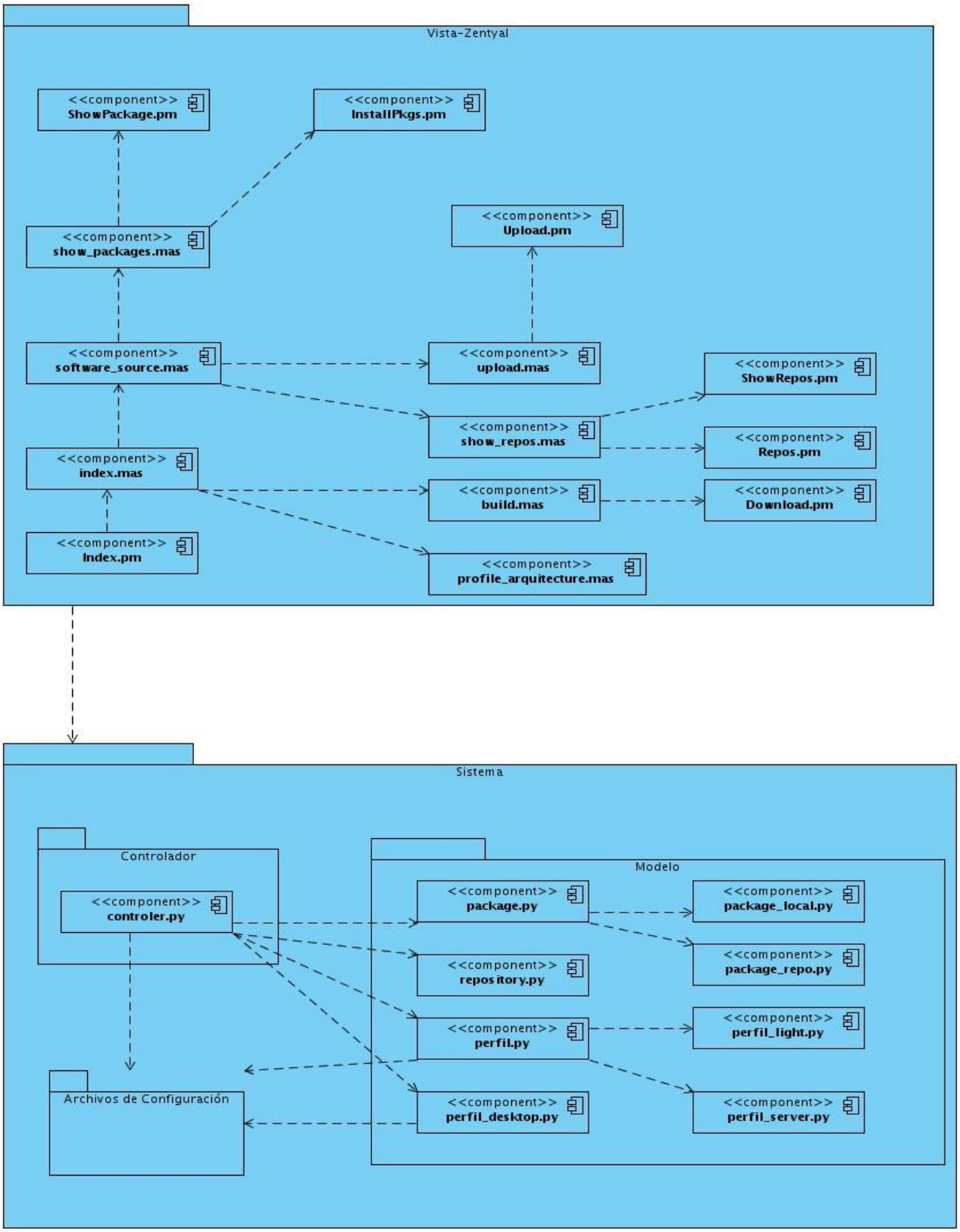


Fig. 10 Diagrama de Componentes.

En el presente capítulo se identificaron los requisitos de software con que constará SCPN y se observó cómo es una de las fases más importante en el ciclo de vida de desarrollo de un software. Se pasó por la fase de Análisis y Diseño, donde se preparan las condiciones para dar paso a la fase de implementación del sistema. Se identificaron los Patrones de Casos de Uso a utilizar: Extensión concreta o Inclusión, CRUD, Múltiples actores. Y se diseñaron los diagramas necesarios para la construcción de un sistema robusto.

Capítulo 3: Implementación y Pruebas.

En el presente capítulo se definen los estándares de codificación a seguir a la hora de programar el SCPN. Además se implementa el sistema y se le realizan las pruebas correspondientes, evaluando el funcionamiento del sistema.

3.1 Estándares de codificación.

Las convenciones de código son un conjunto de reglas a seguir para escribir código fuente uniforme y legible. Estas reglas comprenden cómo se deben utilizar el espaciado, los saltos de línea, cómo se deben nombrar las variables, clases y ficheros y cómo se deben escribir instrucciones específicas del lenguaje de programación.

Las convenciones ayudan a entender el código, por lo que resultan útiles a los desarrolladores en las actividades de mantenimiento a programas escritos por otros o por ellos mismos.

La convención de código utilizada para el lenguaje Python en el desarrollo del **SCPN** son las establecidas en el estándar PEP 08 [28] y para el lenguaje Perl es la establecida en “Curso de Perl” [29].

3.2 Implementación.

Una vez que se hace un Análisis y Diseño del sistema, corresponde la fase de implementación, donde se programa el software siguiendo los lenguajes de programación definidos y cumpliendo con sus respectivos estándares de codificación.

Se pueden ver las principales funciones implementadas en el Anexo #8.

3.3 Pruebas.

Es importante resaltar que los principales objetivo de una prueba son:

1. Encontrar y documentar defectos en el *software*.
2. Una prueba tiene éxito si descubre al menos un error.
3. Asegurar la Calidad del *software*.
4. Validar el cumplimiento de los requisitos [30].

Pruebas Unitarias.

Para asegurar en la medida de lo posible el correcto funcionamiento y la calidad del *software* se suelen utilizar distintos tipos de pruebas, como pueden ser las pruebas unitarias, mediante las que se comprueba el correcto funcionamiento de las unidades lógicas en las que se divide el programa, sin tener en cuenta la interrelación con otras unidades. La solución más extendida para las pruebas

unitarias [31] en el mundo Python es *unittest*, este módulo está incluido en la librería estándar de Python.

Pruebas de Caja Negra.

Las pruebas de caja negra son pruebas de sistema, las mismas están encaminadas a encontrar errores en el *software* previo a la entrega final al usuario. Permite obtener un conjunto de condiciones de entrada que ejerciten completamente los requisitos funcionales del programa. Los cuales a su vez especifican una forma de examinar el sistema, incluyendo la entrada y salida con la que se ha de probar y las condiciones bajo las que ha de hacerse dicha acción [32].

Pruebas de Aceptación.

El objetivo de las pruebas de aceptación es probar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento [33].

3.4 Casos de Pruebas.

Debe verificar:

1. Si el producto satisface los requerimientos del usuario, tal y como se describe en las especificación de los requerimientos.

2. Si el producto se comporta como se desea [34].

Diseño del Caso de prueba.

Nombre del Caso de prueba 1: Gestionar Repositorio.

Descripción: Permite la gestión de repositorio, pudiéndose así adicionar, modificar o eliminar el repositorio deseado.

Condiciones de Ejecución: Para el EC 1.2 Modificar repositorio; debe existir al menos 1 un repositorio.

Escenario	Variable	Respuesta del sistema	Flujo central
EC 1.1 Adicionar repositorio.	new_repo http://novarepo.uci.cu/nova 2011 main restricted universe multiverse http://nova.f10.uci.cu/nova 2011 main	Añade un nuevo repositorio a la lista.	Pinchar en la opción **Añadir repositorio**. El sistema muestra un formulario donde se debe introducir el nuevo repositorio. Luego de introducir los datos correctamente pinchar en Añadir.
EC 1.2 Modificar	mod_repo	Actualiza el repositorio	Seleccionar el repositorio

repositorio.	http://nova.f10.uci.cu/nova 2011 main	modificado.	que se desea modificar. Luego pinchar en la opción **Modificar.** El sistema muestra un formulario donde se introducen los cambios deseados. Pinchar en la opción **Guardar.**
EC 1.3 Eliminar repositorio	N/A	Elimina el repositorio seleccionado de la lista.	Seleccionar el repositorio a eliminar. Pinchar en la opción **Eliminar Repositorio.**

Nombre del Caso de prueba 2: Mostrar perfiles y arquitecturas.

Descripción: Se visualizan los perfiles y arquitecturas disponibles para Nova.

Condiciones de Ejecución: Ninguna.

Escenario	Variable	Respuesta del sistema	Flujo central
EC 1.1 Mostrar perfiles y arquitecturas.	perfiles Desktop Server Light arquitecturas i386 amd64	Muestra los perfiles y arquitecturas disponibles.	Pinchar en la opción **Perfil y Arquitectura.**

Nombre del Caso de prueba 3: Mostrar paquetes disponibles.

Descripción: Se visualizan los paquetes disponibles según los repositorios seleccionados.

Condiciones de Ejecución: Debe existir al menos un repositorio.

Escenario	Variable	Respuesta del sistema	Flujo central
EC 1.1 Mostrar paquetes disponibles.	lista_paquetes gajim vlc python nova-standard serere	Muestra y clasifica en grupos los paquetes disponibles.	Pinchar en la opción **Mostrar paquetes.**

Nombre del Caso de prueba 4: Construir imagen ISO.

Descripción: Construye un imagen ISO de Nova.

Condiciones de Ejecución: Debe haber seleccionar el perfil, arquitectura y los paquetes que serán instalados.

Escenario	Respuesta del sistema	Flujo central
EC 1.1 Construir imagen ISO.	Se construye el ISO y envía un correo de notificación al usuario con el enlace de donde puede descargar el producto.	Pinchar en la opción **Construir ISO.**

Nombre del Caso de prueba 5: Construir discos virtuales.

Descripción: Construye discos virtuales con Nova instalado.

Condiciones de Ejecución: Debe haber seleccionar el perfil, arquitectura y los paquetes que serán instalados.

Escenario	Variable	Respuesta del sistema	Flujo central
EC 1.1 Construir discos virtuales.	discos	Se construye el disco y envía un correo de notificación al usuario con el enlace desde donde puede descargar el producto.	Pinchar en la opción **Construir disco virtual.**
	VDI VMDK IMG		

Nombre del Caso de prueba 6: Adicionar paquetes locales a la instalación.

Descripción: Permite subir paquetes locales desde la computadora cliente al servidor.

Condiciones de Ejecución: Ninguna.

Escenario	Variable	Respuesta del sistema	Flujo central
EC 1.1 Adicionar paquetes locales.	direc_paquete	Se guarda en el servidor el/los paquetes subidos por el usuario.	Pinchar en la opción **Subir paquete.**
	/home/user/paquete.deb		

Resultados de las Pruebas.

Al sistema se le realizaron tres iteraciones de pruebas, en la primera se encontraron 15 no conformidades, en la segunda 5 y en la tercera ninguna no conformidad.

En el presente capítulo se definen los estándares de codificación a seguir, con el objetivo de escribir un código fuente uniforme y legible. Se implementó el SCPN permitiendo crear sistema GNU/Linux Nova a la medida. Además se diseñaron los casos de pruebas y se realizaron las pruebas definidas.

CONCLUSIONES

En la actualidad, Cuba atraviesa por un proceso de migración a software libre, que constituye una necesidad desde el punto de vista económico y legal. El cliente juega un papel importante en dicho proceso, por lo que es necesario facilitarle este.

Como resultado de este trabajo, se cuenta con una propuesta de sistema de construcción de personalizaciones, que permite elaborar soluciones a la medida satisfaciendo así las necesidades de los clientes. Este sistema es de vital importancia para la migración a software libre ya que agilizará el proceso de construcción de personalizaciones en el proyecto Nova.

Se investigaron minuciosamente los sistemas de construcción de personalizaciones UCK, Reconstructor, ISO Master y NIM, demostrando la factibilidad del uso de este último como base para el SCPN.

Por tanto, se concluye que los objetivos propuestos para el presente trabajo de diploma, han sido cumplidos satisfactoriamente.

RECOMENDACIONES

- Poner en práctica el uso de este sistema.
- Agregar nuevas funcionalidades al sistema que permita crear personalizaciones de otras distribuciones GNU/Linux según los objetivos del proyecto y de la migración.

REFERENCIAS BIBLIOGRÁFICAS.

[1] Rolando Alfredo Hernández León, Sayda Coello González, “EL PROCESO DE INVESTIGACIÓN CIENTÍFICA”, Ciudad de La Habana, Editorial Universitaria, 2011, 110,
(Consultada Septiembre, 2011).

[2] Zazzle, “¿Qué significa personalización?”,
[Disponible en: http://zazzle-es.custhelp.com/app/answers/detail/a_id/1138/~/%E2%BFqu%E3%A9-significa-personalizaci%E3%B3n%3F], (Consultada Septiembre, 2011).

[3] “Programa informático”, [Disponible en: http://www.mashpedia.es/Programa_informático], (Consultada Septiembre, 2011).

[4] “Sistema Operativo”,
[Disponible en: http://www.buenastareas.com/ensayos/Sistema-Operativo/3406833.html], (Consultada Septiembre, 2011).

[5] “Repositorios de software”,
[Disponible en:
https://docs.fedoraproject.org/esES/Fedora/14/html/Software_Management_Guide/ch02s02.html] (Consultada Octubre, 2011).

[6] Pablo Castagnino, “Cómo personalizar una imagen de Ubuntu según tus necesidades”,
[Disponible en: http://usemoslinux.blogspot.com/2011/09/como-personalizar-una-imagen-de-ubuntu.html], (Consultada Octubre, 2011).

[7] Pablo Castagnino, “Cómo construir una versión personalizada de Ubuntu”,
[Disponible en: http://usemoslinux.blogspot.com/2010/04/como-construir-una-version.html], (Consultada Noviembre, 2011).

[8] Antonio López, “Iso Master: Editor de imágenes ISO”, 2008,
[Disponible en: http://www.juntadeandalucia.es/averroes/iesgador/spip.php?article38], (Consultada Noviembre, 2011).

[9] “Métodos Ágiles”, 2008,
[Disponible en: http://metodosagiles.blogspot.com/], (Consultada Noviembre, 2011), (Consultada Diciembre, 2011).

[10] “OpenUP como alternativa metodológica para proyectos pequeños de software” , 2008, [Disponible en: http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html], (Consultada Diciembre, 2011).

[11] F. Javier Carazo Gil, “Geany, algo más que un editor de código para Gnome”, 2010,

[Disponible en: http://www.linuxhispano.net/2010/02/02/geany-algo-mas-que-un-editor-de-codigo-para-gnome/], (Consultada Diciembre, 2011).

[12] “Visual Paradigm Suite”,
[Disponible en: http://www.newwebstar.com/programas/67014-visual-paradigm-suite-41-sp1.html], (Consultada Enero, 2012).

[13] “Perl y la Web”, 2007,
[Disponible en: http://www.wikilearning.com/monografia/perl_y_el_web/20838],
(Consultada Enero, 2012),

[14] “¿Que es Python?”,
[Disponible en: http://www.python.com.co/], (Consultada Enero, 2012).

[15] Machtelt Garrels, “*Bash Guide for Beginners*”, 2010,
[Disponible en: http://www.etnassoft.com/biblioteca/bash-guide-for-beginners/],
(Consultada Febrero, 2012).

[16] “Zentyal 2.2 Documentación Oficial”, 2011,
[Disponible en: http://doc.zentyal.org/es/], (Consultada Febrero, 2012), (Consultada Febrero, 2012).

[17] Especificación de Requisitos Software según el estándar de IEEE 830, 2001,
[Disponible en: http://siml.googlecode.com/files/ERS.pdf], (Consultada Marzo, 2012).

[18] “Análisis y Diseño Integral de Software”,
[Disponible en: http://www.funiber.org/areas-de-conocimiento/tecnologias-de-la-informacion/analisis-y-diseno-integral-de-software/], (Consultada Marzo, 2012).

[19] Judith Meles, Elizabeth Jeinson, Cecilia Massano, Patrones de Casos de Uso , 2008,
www.scribd.com/doc/62031003/PatronesDeCasosDeUso, (Consultada Marzo, 2012).

[20] “UNIVERSIDAD DE SAN CARLOS DE GUATEMALA”, 2006,
[Disponible en: http://es.scribd.com/doc/36467270/108/Diagrama-de-clases-del-analisis-de-la-realizacion-del-caso-de-uso], (Consultada Abril, 2012).

[21] Diagrama de secuencia, 2007,
[Disponible: http://www.chuidiang.com/ood/metodologia/diagrama_secuencia.php],
(Consultada Abril, 2012).

[22] ¿Qué es un Patrón de Diseño?,
[Disponible en: http://msdn.microsoft.com/es-es/library/bb972240.aspx],
(Consultada Abril, 2012).

[23] PATRONES GRASP, 2007,
[Disponible en: http://jorgesaavedra.wordpress.com/category/patrones-grasp],

(Consultada Abril, 2012).
[24] “Patrón Modelo-Vista-Controlador",
[Disponible en: http://www.proactiva-calidad.com/java/patrones/mvc.html],
(Consultada Mayo, 2012).
[25] “MVC: ventajas e inconvenientes”, 2009,
[Disponible en: http://pormeterse.wordpress.com/2009/07/16/mvc-ventajas-e-inconvenientes/],
(Consultada Mayo, 2012).
[26] “Diagrama de Clases”,
[Disponible en: http://sistemasdeinformacion2.wikispaces.com/Diagrama+de+Clases],
(Consultada Mayo, 2012).
[27] Javier Macías del Campo, “INGENIERÍA DEL SOFTWARE. UML”,
[Disponible en: http://es.scribd.com/doc/53551175/10/DIAGRAMAS-DE-COMPONENTES],
(Consultada Mayo, 2012).
[28] “Style Guide for Python Code”, 2012,
[Disponible en: http://www.python.org/dev/peps/pep-0008/] (Consultada Mayo, 2012),
(Consultada Mayo, 2012).
[29] Curso de Perl. 2007, [Disponible en: http://www.redantigua.com/curso-perl/capitulo1.html],
(Consultada Mayo, 2012).
[30] “Pruebas”,
[Disponible en: http://merinde.net/index.php?option=com_content&task=view&id=139&Itemid=194],
(Consultada Mayo, 2012).
[31] “Pruebas en Python”,
[Disponible en: http://mundogeek.net/archivos/2008/09/17/pruebas-en-python/],
(Consultada Mayo, 2012).
[32] “Pruebas de Sofware”, 2007,
[Disponible en: http://www.wiziq.com/tutorial/41789-pruebas], (Consultada Mayo, 2012).
[33] “Pruebas de Aceptación”,
[Disponible en: http://www.greensqa.com/portal/soluciones/calidad-de-productos/62-pruebas-de-aceptacion], (Consultada Junio, 2012).
[34] Ing. Daniel Cafferata Salas, “COMO REALIZAR CASOS DE PRUEBA”, 2009,
[Disponible en: http://www.calidadysoftware.com/testing/casos_de_prueba.php],
(Consultada Junio, 2012).
[35] “LENGUAJE DE PROGRAMACIÓN”, 2012,

[Disponible en: http://portafoliowebalejandrab.blogspot.com/2012/02/lenguaje-de-programacion-c.html], (Consultada Junio, 2012).

[36] “Herramientas de Modelado”, 2012,

[Disponible en: http://alegsa.com.ar/Dic/herramienta%20de%20modelado.php], (Consultada Junio, 2012).

BIBLIOGRAFÍA

1. Antonio López, “Iso Master: Editor de imágenes ISO”, 2008, [Disponible en: http://www.juntadeandalucia.es/averroes/iesgador/spip.php?article38]
2. “Análisis y Diseño Integral de Software”, [Disponible en: http://www.funiber.org/areas-de-conocimiento/tecnologias-de-la-informacion/analisis-y-diseno-integral-de-software/]
3. “Bash Reference Manual”, [Disponible en: http://www.gnu.org/software/bash/manual/bashref.html]
4. “Definición de Herramienta”, [Disponible en: http://www.definicionabc.com/general/herramienta.php]
5. Curso de Perl. 2007, [Disponible en: http://www.redantigua.com/curso-perl/capitulo1.html]
6. Disciplinas del OpenUP/Basic, [http://epf.eclipse.org/wikis/openupsp/openup_basic/disciplinegroupings/openup_basic_disciplines,__BZycP1UEdmek8CQTQgrOQ.html]
7. Diagrama de Clases, [Disponible en: http://sistemasdeinformacion2.wikispaces.com/Diagrama+de+Clases]
8. Especificación de Requisitos Software según el estándar de IEEE 830 , 2001, [Disponible en: http://siml.googlecode.com/files/ERS.pdf]
9. F. Javier Carazo Gil, “Geany, algo más que un editor de código para Gnome”, 2010, [Disponible en: http://www.linuxhispano.net/2010/02/02/geany-algo-mas-que-un-editor-de-codigo-para-gnome/]
10. Javier Macías del Campo, “INGENIERÍA DEL SOFTWARE. UML”, [Disponible en: http://es.scribd.com/doc/53551175/10/DIAGRAMAS-DE-COMPONENTES]
11. Linux – Shell, 2008, [Disponible en: http://es.kioskea.net/contents/linux/linshell.php3]
12. Linux Cuba, [Disponible en: http://www.cubasolidarity.net/infomed/www.linux.cu/default.htm]
13. LENGUAJE DE PROGRAMACIÓN, 2012,

[Disponible en: http://portafoliowebalejandrab.blogspot.com/2012/02/lenguaje-de-programacion-c.html]

14. Lucas, ”Soporte de hardware”,
    [Disponible en: http://gluv.univalle.edu.co/MasDoc/Cursos/Avanzado/node146.html]
15. “Métodos Ágiles”, 2008,
    [Disponible en: http://metodosagiles.blogspot.com/]
16. “Herramientas de Modelado”, 2012,
    [Disponible en: http://alegsa.com.ar/Dic/herramienta%20de%20modelado.php]
17. Machtelt Garrels, “Bash Guide for Beginners”, 2010,
    [Disponible en: http://www.etnassoft.com/biblioteca/bash-guide-for-beginners/]
18. “OpenUP como alternativa metodológica para proyectos pequeños de software ”. 2008,
    Disponible en: http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html
19. Pablo Castalio. “Cómo construir una versión personalizada de Ubuntu ”, Disponible en:
    http://usemoslinux.blogspot.com/2010/04/como-construir-una-version.html
20. PATRONES GRASP, 2007,
    [http://jorgesaavedra.wordpress.com/category/patrones-grasp]
21. Patrón "Modelo-Vista-Controlador"
    [Disponible en: http://www.proactiva-calidad.com/java/patrones/mvc.html]
22. Programa informático,
    [Disponible en: http://www.mashpedia.es/Programa_informático]
23. Pablo Castagnino, “Cómo personalizar una imagen de Ubuntu según tus necesidades”,
    [Disponible en: http://usemoslinux.blogspot.com/2011/09/como-personalizar-una-imagen-de-ubuntu.html]
24. Pablo Castagnino, “Cómo construir una versión personalizada de Ubuntu”,
    [Disponible en: http://usemoslinux.blogspot.com/2010/04/como-construir-una-version.html]
25. “Perl y la Web”, 2007, [Disponible en:
    http://www.wikilearning.com/monografia/perl_y_el_web/20838]
26. “Pruebas de Sofware”, 2007,
    [Disponible en: http://www.wiziq.com/tutorial/41789-pruebas]

27. “Pruebas en Python”,
[Disponible en: http://mundogeek.net/archivos/2008/09/17/pruebas-en-python/]
28. “Proceso de personalización paso a paso”, 2010,
[Disponible en: http://msdn.microsoft.com/es-es/library/ms243882.aspx]
29. “Python Documentation”, 2012,
[Disponible en: http://www.python.org/doc/]
30. “Perl Documentation”, 2012,
[Disponible en: http://perldoc.perl.org/]
31. ¿Qué es un Patrón de Diseño?,
[Disponible en: http://msdn.microsoft.com/es-es/library/bb972240.aspx]
32. “¿Que es Python?”,
[Disponible en: http://www.python.com.co/]
33. Repositorios de software, [Disponible en:
https://docs.fedoraproject.org/esES/Fedora/14/html/Software_Management_Guide/ch02s02.html]
34. Sergio Peatonal. “La creación de Linux”. 2007, Disponible en:
http://linuxubuntuandmore.blogspot.com/2007/11/la-creacin-de-linux.html
35. Style Guide for Python Code. 2011,
[Disponible en: http://www.python.org/dev/peps/pep-0008/]
36. SOFTWARE, C. D. A. D. I. D. Flujo de Trabajo de requerimientos. En Tema No 2.Conferencia 4.Flujo de Trabajo de requerimientos. Universidad de las Ciencias Informáticas.Ciudad(espacio) de La Habana.Cuba. Curso 2007-2008. p. 15.
37. Sistema Operativo,[Disponible en: http://www.buenastareas.com/ensayos/Sistema-Operativo/3406833.html]
Ubuntu Customization Kit, 2012,[Disponible en: http://uck.sourceforge.net/]
38. “UNIVERSIDAD DE SAN CARLOS DE GUATEMALA”, 2006,
[Disponible en: http://es.scribd.com/doc/36467270/108/Diagrama-de-clases-del-analisis-de-la-realizacion-del-caso-de-uso]
39. ¿Qué es un Patrón de Diseño?,
[Disponible en: http://msdn.microsoft.com/es-es/library/bb972240.aspx]
40. Zazzle, “¿Qué significa personalización?”,

[Disponible en: http://zazzle-es.custhelp.com/app/answers/detail/a_id/1138/~/%E2%BFqu%E3%A9-significa-personalizaci%E3%B3n%3F],

41. "Zentyal 2.2 Documentación Oficial", 2011, [Disponible en: http://doc.zentyal.org/es/]

ANEXOS

Anexo #1: Entrevistas realizadas.

Entrevista realizada al ingeniero Yunier Soler Franco, desarrollador de la herramienta de personalización **NIM**, la cual sirvió de base para el Sistema de Construcción de Personalizaciones de Nova.

Preguntas realizadas:

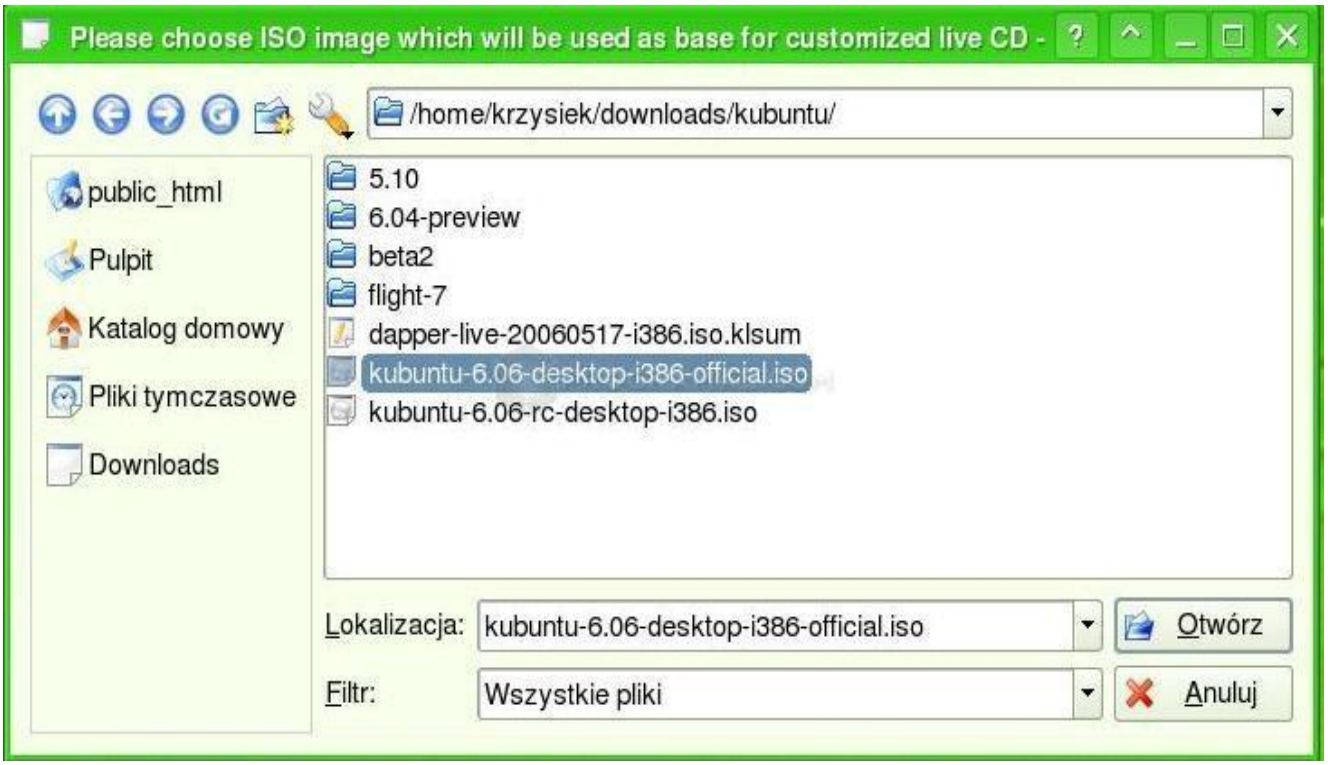
¿Fecha de implantación?

¿Licencia que posee?

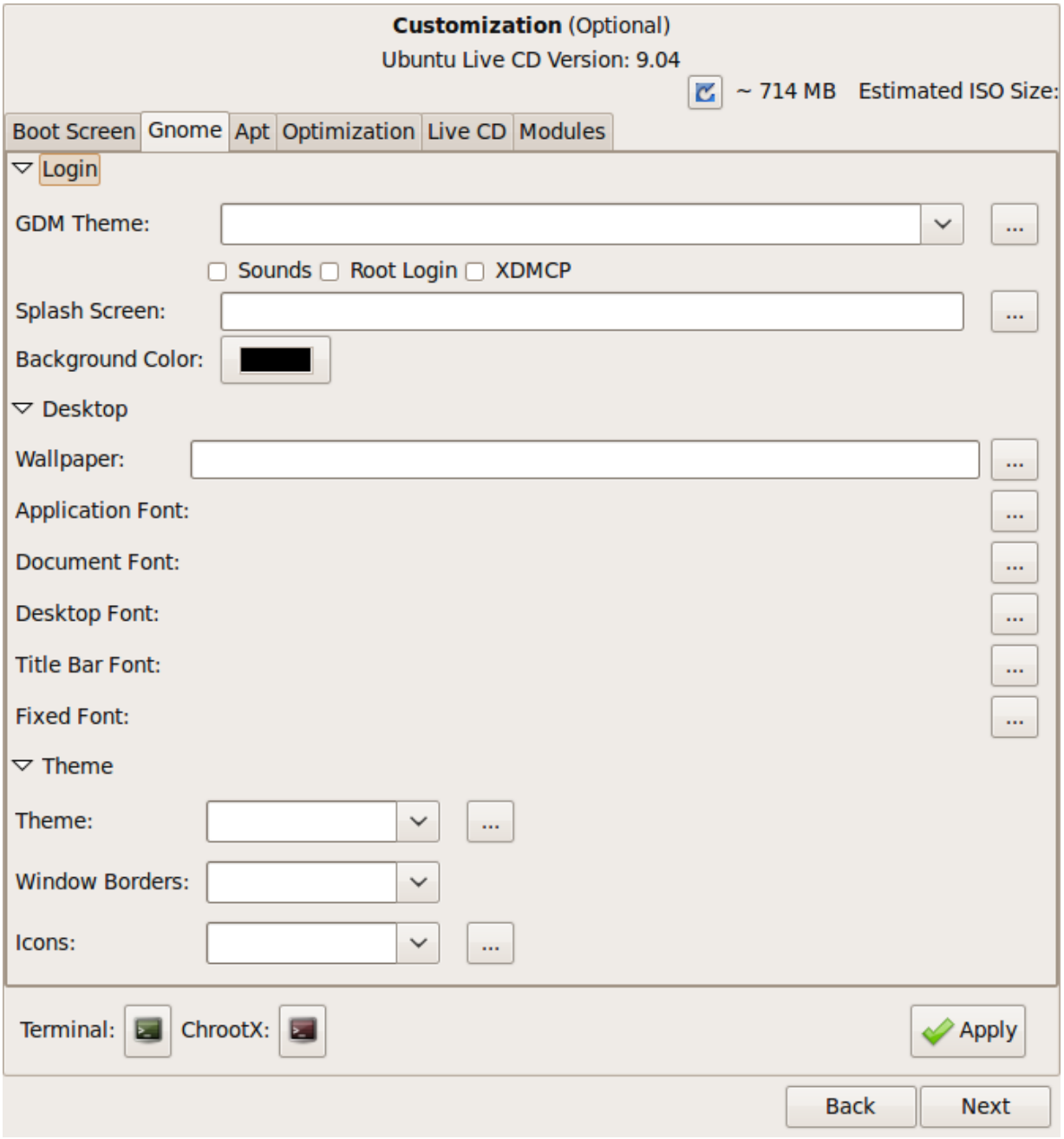
¿Objetivo de la herramienta?

Y preguntas acerca del código.

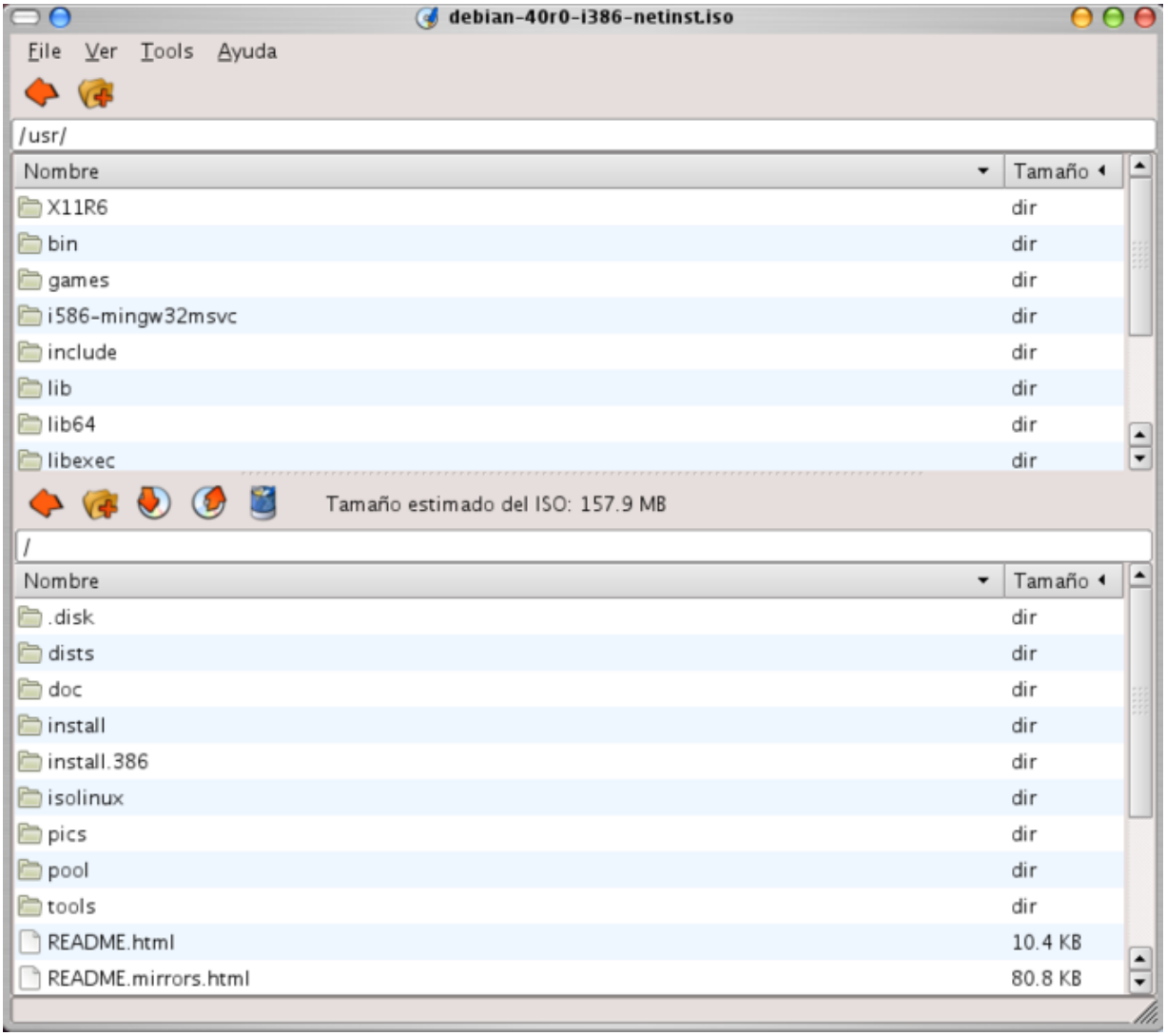
Anexo #2: Herramienta UCK.



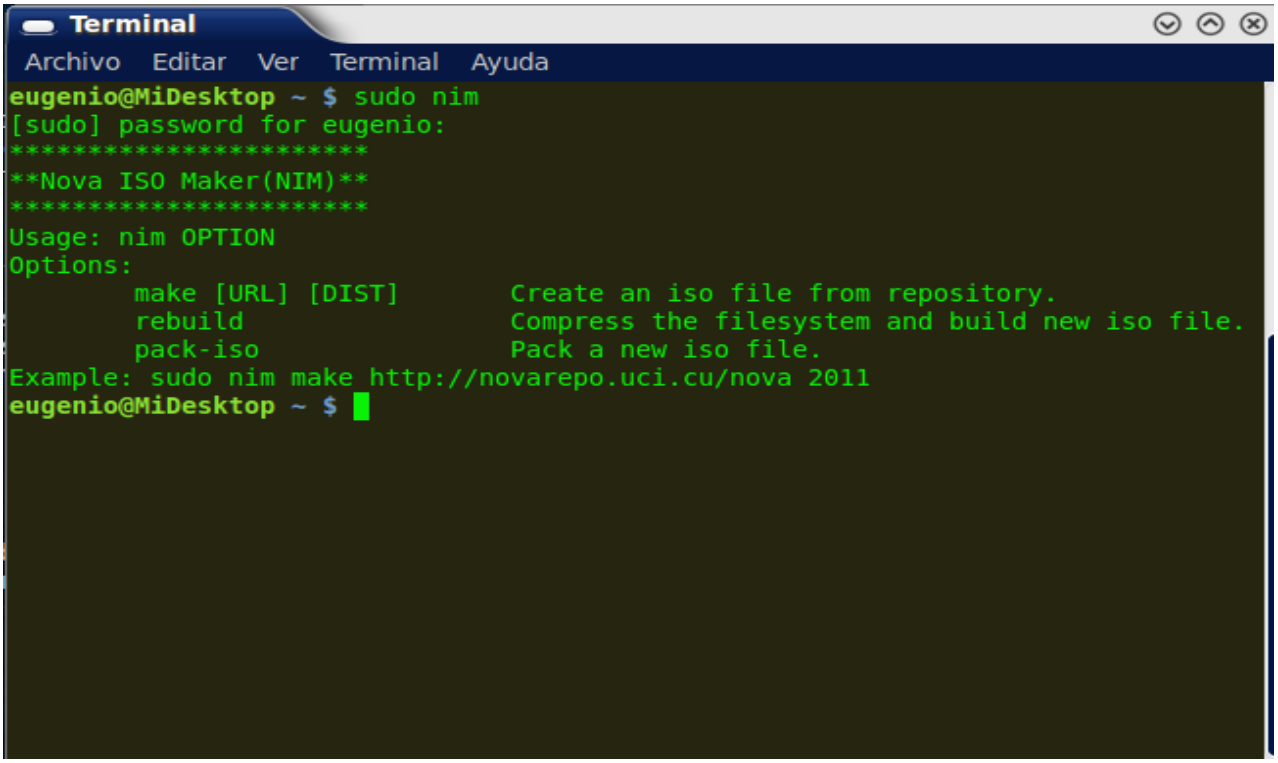
Anexo #3 Herramienta Reconstructor.



Anexo #4: Herramienta ISO Master.



Anexo #5: Herramienta NIM.



Anexo #6: Plataforma Zentyal.

Anexo #7 Módulos de Zentyal.

Anexo #8 Funciones importantes del sistema.

Instalar sistema base para el perfil Nova Desktop.

```
def prepare_for_chroot(self):
        print "Make the chroot environment..."

        print self.root_path+"/ect/apt/sources.list"

        call(["debootstrap", "--components="+self.comp_deboost, "2011", self.root_path,
"http://10.3.10.40/nova"])
        call(["mount", "--bind", "/dev", self.root_path+"/dev"])
        shutil.copy("/etc/hosts", self.root_path+"/etc/")
        shutil.copy("/etc/resolv.conf", self.root_path+"/etc/")

        f = open("../File/config/sources1.list", "r")
        all = f.readlines()
        f.close()
        g = open(self.root_path+"/ect/apt/sources.list", "w")
        for rep in all:
        g.write("deb "+rep.strip()+"\n")
        g.close()

        print "Make a backup copy for /sbin/initctl..."
        call(["chroot", self.root_path, "mv", "/sbin/initctl", "/sbin/initctl.nim_blocked" ])

        print "Prepare the chroot environment..."
        call(["chroot", self.root_path, "mount", "none", "-t", "proc", "/proc" ])
        call(["chroot", self.root_path, "mount", "none", "-t", "sysfs", "/sys" ])
        call(["chroot", self.root_path, "mount", "none", "-t", "devpts", "/dev/pts" ])
        call(["mount", "--bind", "/dev", self.root_path+"/dev"])
        call(["chroot", self.root_path, "apt-get", "update"])
        call(["chroot", self.root_path, "apt-get", "install", "--yes", "dbus"])
        call(["chroot", self.root_path, "apt-get", "install", "--force-yes", "--yes", "sbm"])

        p1 = Popen(["chroot", self.root_path, "dbus-uuidgen" ], stdout=PIPE)
        output = p1.communicate()[0]
        f = open(self.root_path+"/var/lib/dbus/machine-id","w")
        f.write(output)
        f.close
        call(["chroot", self.root_path, "dpkg-divert", "--local", "--rename", "--add", "/sbin/initctl"])
        call(["chroot", self.root_path, "ln", "-s", "/bin/true", "/sbin/initctl"])
```

Instalar sistema base para Nova light / Nova Server.

```
def do_apt_extract(self, profile):
        (exitstatus, ex) = commands.getstatusoutput("cd " + self.imagepath+"; apt-extract \
        -c -p "+ self.imagepath+ " -A " +self.arquitecture+ " -k "+self.private_key+ " -d Nova -s
        2011")

        if exitstatus != 0 :
        exit(ex)
        else :
        # add debootstrap packages
        os.system("cd " + self.imagepath+"; apt-extract -di")

        if profile == "Light":
        os.system("cd " + self.imagepath+"; apt-extract -a nova-light")
        else:
        os.system("cd " + self.imagepath+"; apt-extract -a nova-server")

        os.system("cd " + self.imagepath+"; apt-extract -a grub-pc")

        call(["chmod", "777", "-R", self.imagepath])
        if os.path.isdir(self.imagepath+"/morgue"):
        call(["rm", "-r",self.imagepath+"/morgue"])
```

Glosario de Términos.

Bash: *Bourne Again Shell.*

UCK: *Ubuntu Customization Kit.*

CASE: *Computer Aided Software Engineering.*

CRUD: Creating, Reading, Updating, Deleting.

ERS*:* Especificación de Requisitos Software.

Herramientas de modelado: Son herramientas que se emplean para la creación de modelos de sistemas que ya existen o que se desarrollarán [36].

IDE: ***Integrated Development Environment*** es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien poder utilizarse para varios [35].

Patrón: En la tecnología de objetos es una descripción de un problema y la solución, a la que se le da un nombre, y que se puede aplicar a nuevos contextos

UML*:* *Unified Modeling Language.*