Universidad de las Ciencias Informáticas

Facultad 1

Herramienta para le gestión de los servicios en Nova 4.0.

Trabajo de diploma

para optar por el título de

Ingeniero en Ciencias Informáticas.

Autor:

Johan Travieso Castro.

Tutores:

Ing. Daniel Hernández Bahr.

Ing. Dayron Pérez Roldan.

Ciudad de la Habana,

Junio de 2013.

Declaración de autoría

Declaro ser el autor del presente trabajo de diploma y le ofrezco a la Universidad de las Ciencias Informáticas total autoridad para su uso en beneficio propio.

Para constancia de mi declaración firmo a los ___ del mes de _____ del año 2013.

Johan Travieso Castro

_____

Firma del Autor

Ing. Daniel Hernández Bahr

_____

Firma del Tutor

Ing. Dayron Pérez Roldan.

_____

Firma del Tutor

Dedicatoria

A quien es mi mayor paradigma como profesional y científico. Le dedico mi tesis a mi padrastro y padre de crianza por siempre. Al Dr. Antonio Mayol Alfonso, deslumbras a todos los que te rodeamos con tus conocimientos, eres una meta que muy pocos podrán alcanzar en toda su vida, gracias por estar presente en todo momento.

Agradecimientos

- A la Diosa de mi vida y luz de mi camino, mi madre, por dedicarme tu vida sin pedir nada a cambio, por estudiar la mía para poder serme útil en cada etapa, por ser la fuente de amor y ternura más grande que he conocido y que nada superará jamás, por ser el mayor orgullo del que presumo cada día, gracias por existir.
- A mi padre por estar presente en la distancia, por todos sus consejos, por todo su apoyo.
- A mi familia en la isla, con ella me crié, crecí y con ellos están mis raíces, a mis abuelos por todo lo vivido, a mis tías que son especiales, a mis primos, a mis tíos, a todos.
- A mi familia de Caimito, la universidad no solamente me ofreció una carrera y un título, nos dio la oportunidad de estar un tiempo juntos, les agradezco a todos por estar al tanto de mí, por su ayuda, por su cariño y por cuidarme. A mis abuelos paternos que son un manojo de afectos y atenciones, a mis tías y a mis tíos, a mis primas y a mis primos, a Diris por secuestrarme.
- Al maestro Daniel, por ser más que un tutor, por ser un amigo y el brazo derecho que me ayudó en esta tesis, no hubiera sido posible sin él.
- A mi oponente por su disponibilidad y su apoyo, fue un tutor más en este trabajo.
- A todos mis amigos de la universidad, por todos los momentos, por los buenos por los malos, agradezco la oportunidad de haberlos conocido a todos, a Oscar, a Andy a Celso, a Yenny, a Elizabeth, a Leodán, a todos.

Resumen

En el presente trabajo se muestra el desarrollo de una aplicación que facilita la interacción y mejora la comunicación de los usuarios con la gestión de los servicios en la *distribución* cubana de GNU/Linux Nova 4.0.
La información acerca de la investigación se encuentra documentada con el objetivo de arribar a un entendimiento del por qué la necesidad de ofrecer a los usuarios aplicaciones con interfaces gráficas de usuario.
Se realiza además un estudio del estado del arte de las aplicaciones existentes que ofrecen interfaces gráficas de usuario para la interacción con los servicios en GNU/Linux y se fundamenta la razón por la cual es necesario mejorar la interacción de estas interfaces.

Palabras clave

Servicios, demonios, interfaces gráficas de usuario.

Abstract

The present document shows the development of an application which makes easier the interaction and improves the communication of software users with the management of the services in the Cuban GNU/Linux distribution Nova 4.0.
The information about the investigation is documented with the objective of accomplishing an understanding about the need of offering applications with graphical user interfaces with a friendly state to users.
Moreover a study of the state of the art about the applications available which offer graphical user interfaces for the interaction with services in GNU/Linux is carried out and are exposed the reasons about why it is necessary to improve the interaction with those interfaces.

Keywords

Services, daemons, graphical user interfaces.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Necesidad

Las interfaces gráficas de usuario (GUI por sus siglas en inglés) son el gancho principal para la atracción de usuarios hacia los productos de las compañías de software. Las GUI surgen con el objetivo principal de introducir cada vez más el software en el mundo del negocio, la investigación y el consumo e incrementar así el uso de las computadoras por usuarios en general. Estas interfaces son el paso siguiente de los modos de comunicación e interacción existentes en las computadoras por medio de interfaces en modo de terminal en las cuales se hace tedioso el trabajo pues consisten de una interfaz en modo texto a la cual se le introducen órdenes y se obtiene información.

En un escenario genérico de comunicación existen tres entidades básicas. Un emisor que envía la información, un medio de transmisión por el cual viaja la información codificada y un receptor que será capaz de entender esta información. En el escenario de comunicación visual en el entorno de la informática contamos con una computadora u ordenador, una interfaz y un usuario, de modo que las entidades extremo interactúan ofreciendo y recibiendo información recíprocamente por medio de una interfaz. Las GUI mejoran la comunicación con los usuarios por medio de imágenes, controles y texto con respecto a las interfaces en modo de terminal porque estas últimas ofrecen la misma información en detrimento de la interacción. (1)

Antecedentes

Las primeras apariciones de GUI datan del período comprendido entre los años 1970 – 1981, etapa en la que se empezó a trabajar en el desarrollo de dicho modelo interactivo y se formalizarían los primeros modelos de ordenadores con elementos *multimedia*, capaces de representar información textual y visual dando la posibilidad de interactuar de forma más intuitiva. Como protagonista y pionero en esta rama tenemos al centro de investigaciones PARC (Palo Alto Research Center) una división que la empresa Xerox Corporation destinó a la investigación y el desarrollo de la comunicación e interacción entre las personas y los ordenadores. Ya para el año 1981 se realizaría el lanzamiento del primer ordenador que cumpliera con este modelo resumiendo once largos años de investigación. Esta primera computadora personal no fue un producto comercial y su uso estuvo orientado a centros de investigación y universidades. (1)

Sucediendo a la empresa Xerox Corporation comenzó una larga carrera en el desarrollo de las interfaces gráficas de usuario para el uso de ordenadores como fueron los casos de Apple con

sus entornos Lisa y Macintosh (1984), Digital Research con Next GEM (1985), en ese mismo año la empresa Commodore lanzó una computadora con una GUI llamada WorkBench y aparece la versión 1.0 de Microsoft Windows de mano de la empresa IBM. (1)

El mundo del software libre no se quedó atrás. Evolucionando desde una interfaz en modo de terminal pero con muchas más funcionalidades que los sistemas anteriormente mencionados. Aparecen los *entornos de escritorios* KDE y GNOME. Estos dos entornos de escritorios han sido los más usados durante mucho tiempo incluso en la actualidad mantienen su competencia (1). Unido a la introducción de los entornos de escritorio se comenzaron a desarrollar una amplia gama de aplicaciones para los usuarios de todo tipo desde los más avanzados hasta los básicos y se extendió el uso del software libre a todas las partes del mundo. Estos entornos de escritorio han sido desarrollados con la intención de ser usados en sistemas operativos GNU/Linux. En estos sistemas operativos se cuenta con una amplia gama de aplicaciones y procesos de configuración del sistema operativo que brindan mucha potencia para la administración del sistema. Uno de los casos es la gestión de los servicios, que se realiza introduciendo comandos en una terminal o moviendo los enlaces simbólicos de los archivos de configuración, dicho caso será el tema a tratar en el presente trabajo de diploma. En nuestra Universidad se desarrolla y mantiene una distribución de GNU/Linux llamada Nova, que en estos momentos se encuentra en su versión 4.0 y carece de una interfaz gráfica de usuario para la gestión de los servicios de forma nativa, o sea, que venga incluida con el sistema operativo.

Por lo anteriormente expuesto, se plantea lo siguiente:

Situación Problémica

Actualmente en la distribución cubana de GNU/Linux, Nova 4.0, no se cuenta con una interfaz gráfica de usuario para interactuar con la gestión de los servicios, haciéndose el proceso por medio de una interfaz en modo terminal introduciéndosele órdenes o editando los archivos de configuración pertinentes, siendo este modo de interacción más tedioso que si se contara con una interfaz gráfica de usuario.

Problema Científico

¿Cómo mejorar la interacción con la gestión de los servicios para Nova 4.0 de forma nativa por medio de una interfaz gráfica de usuario?

Objetivo General

Desarrollar una herramienta con interfaz gráfica de usuario que mejore la interacción con la gestión de los servicios en GNU/Linux Nova 4.0.

Objetivos Específicos

1. Estudiar de la gestión de los servicios en GNU/Linux.
2. Analizar de la estructura de los niveles de ejecución en GNU/Linux.
3. Desarrollar de una herramienta que permita:

   a) Gestionar los estados de ejecución de los servicios en GNU/Linux Nova 4.0.

   b) Habilitar o deshabilitar el inicio automático de un servicio deseado.

Objeto de Estudio

El objeto de estudio de este trabajo lo constituye la gestión de los servicios en GNU/Linux.

Campo de acción

El campo de acción del trabajo abarca la gestión de los servicios en la distribución cubana de GNU/Linux Nova 4.0.

Idea a Defender

La implementación una aplicación que medie como interfaz gráfica entre la gestión de los servicios en GNU/Linux Nova 4.0 y los usuarios mejorará la comunicación e interacción con la administración de la distribución en cuanto a usabilidad y aceptación por parte de éstos.

Métodos de Investigación

1. Análisis histórico lógico, permite realizar un estudio para analizar cómo ha sido la evolución de las interfaces gráficas de usuario y sus antecedentes, revelando las etapas principales de su desarrollo.
2. Observación, permite analizar cada fase del proceso y saber si se está realizando cada tarea correctamente, tomando experiencia para aplicarla a tareas futuras.
3. Experimental, se ve reflejado a la hora de realizar las pruebas luego de terminada su implementación para verificar el correcto funcionamiento de la misma.

Tareas de Investigación

1. Estudio de la documentación existente acerca del marco teórico que abarca el trabajo para lograr un entendimiento acerca del funcionamiento de los servicios en GNU/Linux.
2. Investigación acerca de la gestión de los servicios en GNU/Linux y Nova 4.0 esclarecerá la forma de interactuar con los mismos.
3. Recopilación de información acerca del estado del arte en cuanto a las herramientas

existentes en el ámbito del trabajo permitirá establecer comparaciones y establecerá las bases necesarias para la materialización del trabajo.

4. Estudiar las metodologías de desarrollo y el leguaje de modelado apropiados ofrecerá una guía y organización para el diseño de la aplicación.
5. Elaboración del modelo del sistema, permitirá detectar las clases principales que se utilizarán en el sistema.
6. Definición de los requisitos funcionales y no funcionales conllevará a que la aplicación se ejecute correctamente y se establecerán las prestaciones mínimas para su uso.
7. Aplicación de patrones de diseño a las clases del sistema para optimizar las responsabilidades y comportamiento de las mismas.
8. Realización de pruebas a la aplicación para validar que las funcionalidades se ejecuten correctamente.

Al concluir el presente trabajo se espera como posible resultado:

- Una herramienta con interfaz gráfica de usuario para la gestión de servicios en la distribución GNU/Linux Nova 4.0.
- Incrementar la interacción con la administración del sistema en GNU/Linux Nova 4.0.

Capítulo 1. Fundamentación teórica: Se explican conceptos y el funcionamiento de los servicios en GNU/Linux, GNU/Linux Nova 4.0 y sistemas homólogos. Además se realiza un estudio del estado del arte de las aplicaciones existentes con GUI orientadas a la gestión de los servicios.

Capítulo 2. Diseño de la propuesta de solución: Se describe el diseño de la solución en pos de ofrecer interfaces gráficas de usuario para la gestión de los servicios Nova 4.0.

Capítulo 3. Implementación y solución de pruebas: Se realizan pruebas de funcionamientos y aceptación a la aplicación desarrollada y se muestran los resultados obtenidos de las mismas.

Capítulo 1. Fundamentación teórica

Introducción

En este capítulo se abarcará el tema de la gestión de los servicios en GNU/Linux, en la distribución Nova así como sistemas homólogos. Se explican además conceptos acerca de la estructura de los mismos, se hace una propuesta de solución y se exponen las tecnologías a utilizar.

1.1 Servicios o demonios

Los servicios o demonios en GNU/Linux son procesos especiales que se ejecutan en un segundo plano de forma independiente. Esto significa que no están en interacción directa con el usuario y que sus acciones se ejecutan según eventos u ocurrencias específicas. Estas ocurrencias pueden ser un intervalo especificado de tiempo, el depósito de un archivo en un directorio específico, la recepción de un correo o una solicitud web hecha a través de una comunicación en particular. (2)

El término demonio fue usado por primera vez en el proyecto MAC por sus programadores. Estos se inspiraron en el demonio imaginario de Maxwell en termodinámica que lo ayudaba a ordenar moléculas de forma incansable en un segundo plano. El término fue usado luego para describir procesos que trabajan en segundo plano para llevar a cabo las tareas del sistema. El primer demonio empleado en una computadora fue un programa que hacía respaldos en cintas de manera automática, luego de que el término fuera adoptado para el uso en la computación fue racionalizado por el acrónimo DAEMON (Disk And Execution MONitor). (2)

Es válido aclarar que no todos los servicios que se encuentran en el sistema son para la interacción con el usuario. Algunos de estos servicios son conocidos como servicios del tipo *one shot* que son servicios que utiliza el sistema para el arranque o apagado.

1.2 Estructura de los servicios en GNU/Linux

GNU/Linux se torna realmente útil y poderoso debido a la variedad de servicios y servidores que ofrece. Al igual que como se menciona en el epígrafe 1.1 de forma genérica en Linux los servicios se ejecutan sin la intervención del usuario o administrador del sistema. En este caso se utiliza el término de *standalone* o procesos independientes puesto que sus estados se mantienen activos o inactivos indefinidamente a menos que sea emitida una orden o comando para alterarlos. Estos comandos pueden iniciar o detener manualmente un servicio indicado, así para estas acciones existen un conjunto de órdenes. (3)

En la mayoría de los casos con solamente instalar un paquete de software es suficiente para

activar el servicio correspondiente, sin embargo en ocasiones se desea o necesita cambiar el comportamiento de un servicio en específico lo que se traduce casi siempre en modificar la forma en que este inicia.

En el momento en que el *kernel* inicia, una de las últimas acciones que realiza es arrancar el proceso *init*. La función principal de init es iniciar todos los servicios en el momento preciso; este momento está en dependencia de las tareas que se estén realizando, por ejemplo si se están realizando tareas de corte administrativo puede que se desee la no ejecución de ciertos servicios. Para personalizar la ejecución de los servicios al inicio del sistema GNU/Linux ofrece un mecanismo llamado *runlevels* o niveles de ejecución para hacer esto automáticamente. (4)

1.2.1 Init

Finalizando el transcurso de inicio del sistema operativo, el proceso init es lanzado y se le asigna el identificador de proceso (PID por sus siglas en inglés) “1”. Init se encarga de iniciar el sistema de procesos según los niveles de ejecución definidos en el archivo /etc/inittab y se convierte en el “padre” de todos los demonios. Es encargado de chequear también en el archivo cuál es el nivel de ejecución por defecto y en caso de que no se haya establecido ofrecer al usuario una interfaz en modo de terminal para que especifique uno para el inicio en cuestión del sistema operativo. En el momento de apagado init controla la secuencia de detención de los servicios como la última de sus acciones a realizar. (2)

1.2.2 Niveles de ejecución (runlevels)

Los servicios pueden ser gestionados manualmente mediante órdenes de líneas de comando con sus respectivos argumentos y el nombre del servicio que se quiera gestionar. Ahora bien, puede darse el caso de que algún usuario o administrador necesiten una configuración más específica de qué servicios iniciar o detener al inicio del sistema. Para evitar un trabajo redundante cada vez que se inicia el sistema, se ofrece un mecanismo llamado niveles de ejecución para decidir qué servicio inicia o no con el sistema operativo. La mayoría de los usuarios están familiarizados con dos niveles de ejecución. El primero que es el más comúnmente usado es al que se refiere a un modo multiusuario. Este es donde se habilita la autenticación, la red está activa y el sistema se comporta de forma “normal”. El otro nivel de ejecución es el de mantenimiento del sistema o modo mono usuario, donde un solo usuario está autenticado sobre el sistema probablemente haciendo tareas de chequeo o mantenimiento. (3)

En Linux a diferencia de otros sistemas operativos es posible configurar la ejecución de los

servicios en un nivel de ejecución deseado. Por lo general se tienen 7 niveles que inician o detienen los servicios.

0 Detener o apagar el sistema

1 Modo mono usuario, se utiliza mayormente para mantenimiento del sistema,

2 Modo multiusuario, sin soporte de red,

3 Modo multiusuario completo, con servicios de red,

4 No se usa, puede ser empleado para un inicio personalizado,

5 Modo multiusuario completo con inicio gráfico

6 Modo de reinicio.

El fichero /etc/inittab contiene el nivel de ejecución por defecto con que se iniciará nuestro sistema operativo, he aquí una razón más para no dejar una terminal con privilegios de administración abierta pues si se establece como nivel de ejecución al 6 nos encontraríamos frente a un sistema operativo que se reinicia indefinidamente. Además de esto se cuenta con el comando init al cual se le pasa como argumento el nivel de ejecución al que se quiera cambiar de un momento a otro para alguna tarea en específico sin necesidad de editar el archivo /etc/inittab. También se puede usar este comando para apagar o reiniciar pasándole de argumento 0 ó 6 respectivamente, de hecho algunos comandos como *shutdown* o *halt* utilizan el comando init para estas tareas además de otras como sincronización para los respaldos del disco duro.

1.2.3 Niveles de ejecución en Nova 4.0

En Nova 4.0 hay una ligera modificación en cuanto a que aparecen niveles de ejecución más específicos quedando de la forma siguiente:

N Arranque del sistema (NONE)

S Modo mono usuario (no para ser cambiado directamente)

0 Apagado

1 Modo mono usuario

2 – 5 Modo multiusuario

6 Reinicio del sistema

Existen varios estándares para la configuración de los servicios en GNU/Linux como *System-*

V, Upstart o *Systemd*. Estos consisten en definir el sistema de ficheros a utilizar, los tipos de ficheros y los modos en que se ejecutarán. En el directorio /etc/ se encuentran diferentes subdirectorios rc#.d (donde # va desde 0 a 6 o puede ser S) que contienen enlaces simbólicos al directorio /etc/init o /etc/init.d dependiendo de si el estándar de configuración es System-V o Upstart (más adelante se hablará acerca de estos sistemas de procesos) que son los que posee Nova 4.0. Los enlaces simbólicos tienen la siguiente nomenclatura, S<número de dos dígitos><nombre del servicio> o K<número de dos dígitos><nombre del servicio>. Los scripts que tienen la S delante son los que se inician junto con en el nivel de ejecución correspondiente al directorio donde se encuentren y los que tienen la K delante son los que no se inician a no ser emitida una orden. El número que está después de las letras es consecutivo y está entre 01 y 99 y es el orden de prioridad en que se ejecutan los scripts. (3)

El cambio de nivel de ejecución de un servicio puede hacerse eliminando el enlace simbólico del servicio o haciéndolo uno al directorio rc del nivel de ejecución deseado.

1.3 System-V

Algunas distribuciones utilizan System-V como un estándar de una serie de scripts ejecutados. System-V es un diseño de ejecución de scripts. Los scripts de los servicios que maneja System-V están ubicados en /etc/init.d. (2)

1.4 Upstart

Upstart es un sistema de reemplazo basado en eventos para el demonio /sbin/init el cual maneja el inicio de las tareas durante el arranque, las detiene durante el apagado y las supervisa mientras el sistema está ejecutándose. (5)

Características más importantes:

- Tareas y servicios son iniciados y detenidos por eventos.
- Los eventos son generados como tareas y los servicios son iniciados y detenidos.
- Los eventos pueden ser recibidos por cualquier otro proceso del sistema.

Los servicios pueden ser regenerados si “mueren” inesperadamente.

1.5 Herramientas existentes

Para el desarrollo de la aplicación se realiza un estudio del estado del arte actual en pos de justificar la implementación de la misma, a continuación, se describen las herramientas que brindan interfaces gráficas de usuario para la interacción con la gestión de los servicios en GNU/Linux en general.

service

El comando service se utiliza para consultar o modificar los estados de ejecución de los scripts de servicios de System V o un evento de Upstart. Este comando permite además consultar el estado de un servicio o de todos los servicios en el sistema. Como se explica, el hecho de que sea un comando arroja como condición que para interactuar con éste se le introduzcan órdenes a una terminal constituyendo parte de la situación Problémica del presente trabajo de diploma. Por otra parte carece de mecanismos para encargarse de la gestión de los niveles de ejecución quedando incompleta como solución para la gestión de los servicios. (6)

rcconf

Es una herramienta de interfaz de línea de comando para la configuración de los niveles de ejecución que permite controlar cuáles servicios son iniciados cuando el sistema arranca o cuando reinicia. Muestra un menú de todos los servicios y los que ya inician aparecen marcados. Esta herramienta, por el contrario del comando service, carece de funcionalidades para modificar los estados de ejecución de los servicios, siendo otro ejemplo de herramienta incompleta en el contexto actual. (7)

sysv-rc-conf

Ofrece una interfaz en modo de línea de comando fácil de usar para la gestión de los directorios /etc/rc#.d (como se menciona en el epígrafe 1.2.3) y los enlaces simbólicos a los mismos. La interfaz viene en dos modos, uno para permitir iniciar o detener servicios al inicio y otro para acciones más personalizadas de los enlaces simbólicos. Es una herramienta pensada para reemplazar a rcconf pero sin adicionarle más funcionalidades aparte de las referentes a las de los niveles de ejecución quedando incompleta también respecto a la interacción con los estados de ejecución de los servicios. (8)

Boot-up Manager

Boot-up Manager es una aplicación escrita en Perl con Gtk2 para manipular la configuración de los niveles de ejecución de cualquier distribución similar a Debian. Con este software el usuario puede iniciar o detener los servicios sin tener que lidiar con la creación de enlaces simbólicos o escribir comandos desde la consola. En la primera instancia de Boot-up Manager, si se inicia sin los privilegios de súper usuario (root), este muestra un cartel de excepción que indica que se debe iniciar como root para poder acceder a la aplicación. Cuando se inicia Boot-up Manager como root se muestra una vista con una lista de los servicios disponibles en el sistema y una breve descripción de los mismos. Boot-up Manager ofrece la facilidad de interactuar con la gestión de los servicios desde una aplicación con interfaz gráfica de usuario,

ofrece la posibilidad de detener, iniciar, habilitar o deshabilitar los servicios (9). Por otra parte la simbología de los estados es ambigua, dejando margen a la confusión puesto que para ejecutar las acciones hay que hacer click derecho, además no hay leyenda de los estados en el menú ayuda del programa ni en ninguna otra parte de la aplicación. El hecho de que Boot-up Manager se inicie bajo el otorgamiento de los privilegios de súper usuario constituye una mala práctica de programación puesto que las bibliotecas de componentes gráficos pueden tener fallas de seguridad por errores de programación principalmente (10) y conllevar a una intrusión o ataque a algún componente del sistema. La buena práctica consiste en concederle los privilegios de súper usuario a las acciones que se realizan dentro de la aplicación. El software ofrece además una vista avanzada en la que se muestra mucha información acerca de la gestión de los servicios y los niveles de ejecución. Esto para un usuario que solamente necesite saber qué servicios tiene instalados en su máquina, cuáles quiere detener, iniciar o reiniciar y cuáles habilitar o deshabilitar al inicio del sistema y no posea al menos conocimientos medios del tema en cuestión puede perderse ante el arrojo de tanta información.

1.6 Propuesta de solución

Con la intención de cumplir con el objetivo general del presente trabajo de diploma se propone como solución el diseño e implementación de una aplicación con interfaz gráfica de usuario para interactuar con la gestión de los servicios en GNU/Linux Nova 4.0. La aplicación final brindará la posibilidad de modificar los estados y los niveles de ejecución de los servicios a los usuarios que la utilicen. Las acciones que requieran interacción con los servicios se ejecutarán bajo los privilegios establecidos por la aplicación y a los usuarios comprendidos. Con esta propuesta se mejorará la usabilidad y aceptación de la distribución cubana de GNU/Linux Nova 4.0 en relación a la interacción con la gestión de los servicios.

1.7 Herramientas CASE

Es el acrónimo de Computer Aided Software Engineering (Ingeniería de software asistida por computadora). Las herramientas CASE son un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores. Proporcionan la automatización de todo el ciclo de vida de desarrollo de un software, completamente o en alguna de sus fases. Cubren todos los pasos que se pueden dar en el proceso de desarrollo de software y actividades generales que se aplican a lo largo del proceso. (11)

1.7.1 Visual Paradigm

Visual Paradigm (VP) es una poderosa herramienta CASE multiplataforma para el modelado de software. VP le proporciona a los desarrolladores la plataforma de desarrollo vanguardia para desarrollar aplicaciones con calidad rápidamente, mejores, en menos tiempo y a un menor costo. Facilita una interoperabilidad excelente con otras herramientas CASE y los IDEs más utilizados. Se distingue además por su completo desarrollo en modelo-código-despliegue. (11)

1.8 Python

Es un lenguaje de programación multiplataforma y multiparadigma, lo primero indica que su código puede ser ejecutado desde cualquier sistema operativo y lo segundo que soporta varios paradigmas de programación como orientado a objetos, programación funcional y programación estructurada. Es un lenguaje de programación de alto nivel y es interpretado, posee una variedad de estructuras de datos que facilitan la manipulación y organización de la información. Su filosofía hace hincapié en la escritura de un código legible y organizado, estableciendo como condición que tiene que estar indentado, lo cual facilita la lectura e interpretación de otros desarrolladores y mantenedores (12). La condición de ser un lenguaje interpretado no influye mucho en el rendimiento de las aplicaciones que se proponen desarrollar en el presente trabajo puesto que serán aplicaciones que no estarán ejecutándose constantemente sino que serán de uso parcial para y personalizaciones y configuraciones específicas. Más para las ventajas, Python posee una multitud de bibliotecas adaptadas para la interacción con la terminal de los sistemas operativos, permitiendo utilizar las entradas y salidas de los comandos empleados en las configuraciones para la gestión de los en Nova 4.0.

1.9 Gtk, biblioteca de componentes gráficos

Gtk es una biblioteca para crear interfaces gráficas de usuario, trabaja sobre varias plataformas, las de distribuciones GNU/Linux, Unix, Windows entre otras. Posee una arquitectura orientada a objetos basada en C que permite una máxima flexibilidad. Se le han incluido *bindings* para otros lenguajes como C++, Objective-C, Perl, Python, Free Pascal entre otros (13). La razón de por qué usar Gtk y no Qt como biblioteca de componentes gráficos es que Gtk viene instalada por defecto en Nova 4.0.

1.10 Entorno de desarrollo

Como entorno de desarrollo se detectaron dos herramientas a utilizar. Geany para la parte del código y Glade para el diseño de las interfaces.

1.10.1 Geany

Geany es un pequeño y rápido editor de textos con características básicas de un entorno de desarrollo integrado, está disponible para varios sistemas operativos como las distribuciones de GNU/Linux además de que se puede usar en Windows y Mac. Se encuentra distribuido como software libre bajo la licencia pública general de GNU. Tiene soporte para varios lenguajes como, C++, Java, PHP, Python, HTML, Ruby, DocBook, Perl entre otros. (14)

Algunas de sus características son:

- Resaltado de la sintaxis.
- Completamiento de código.
- Mezclado de código.
- Etiquetado de símbolos.
- Salida de terminal incorporada.

Otras más conocidas son:

- Compatible con la mayoría de los lenguajes.
- Varios paneles para acceder mejor a los datos.
- Herramientas para compilar.
- Buscador integrado.

1.10.2 Glade

Es una herramienta de desarrollo para generar componentes gráficos para la construcción de interfaces gráficas de usuarios empleando la librería de Gtk. Independiente del lenguaje de programación, no genera código fuente sino un archivo XML. Usando el objeto GtkBuilder dependiendo del binding del lenguaje de programación que se utilizará como núcleo se pueden importar los componentes generados en el XML. Permite el diseño de una interfaz gráfica desde una interfaz gráfica en vez de escribir un programa usando funciones o librerías de clases lo que le ahora muchísimo tiempo al desarrollador. Está soportada la creación de varios *widgets* gráficos Gtk, permitiendo al programador concentrarse en la implementación de aplicaciones funcionalmente en vez de en su presentación. (15)

1.11 Polkit

Polkit es un conjunto de herramientas a nivel de aplicación para definir y manipular las políticas que permiten a procesos sin privilegios comunicarse con procesos con privilegios en

sistemas operativos similares a Unix. (16)

1.12 D-Bus

D-Bus como abreviatura de Desktop Bus es un sistema de mensajes de bus, una vía simple para que las aplicaciones se comuniquen entre sí. Es un mecanismo de comunicación entre procesos que consta de tres partes (17):

- Una biblioteca libdbus que permite a dos aplicaciones conectarse e intercambiar información.

- Un demonio ejecutable que funciona como bus de mensajes, construido sobre libdbus, al cual pueden conectarse varias aplicaciones. El demonio puede encaminar mensajes desde una aplicación a ninguna o a más aplicaciones.

Bibliotecas adaptadas para su uso en *frameworks* concretos.

1.13 Metodología de desarrollo de software

El uso de las metodologías en el desarrollo de software es una buena práctica que garantiza la correcta creación del producto. El uso y selección de una metodología es un trabajo minucioso, dependiendo de las características del software que se vaya a construir y teniendo en cuenta las ventajas que más se ajustan a las necesidades del equipo de desarrollo.

El objetivo del uso de una metodología es trazar una guía a los desarrolladores para facilitarle la organización y el entendimiento de la implementación, pero los requisitos de un software pueden encontrarse en una muy amplia gama, denotando así la existencia de varias metodologías.

1.14 Open Up

Open Up es una metodología ágil que aplica un enfoque iterativo e incremental dentro de un ciclo de vida estructurado y contiene un conjunto mínimo de prácticas que ayudan al equipo a ser más efectivo en el desarrollo de software. Open Up abraza una filosofía de desarrollo que se basa en la naturaleza colaborativa del desarrollo de software. Se trata de herramientas neutrales que pueden extenderse para alcanzar una amplia variedad de tipos de proyectos.

Es un modelo de desarrollo de software, parte del modelo de Eclipse (Eclipse Process Framework), desarrollado por la fundación Eclipse. Surge como un subconjunto de la metodología RUP (Rational Unified Process) y está basada en la misma reduciéndose a una serie de prácticas para brindarle al equipo de desarrollo de software una metodología con la cual se puede desarrollar de forma eficiente incluyendo en el interior de su proceso el contenido más útil para el desarrollo de un producto de software. (18)

Los principios fundamentales de Open Up son:

- Promover la colaboración para alinear los intereses comunes y difundir el conocimiento.
- Balanceo de las prioridades competitivas para maximizar el valor de los participantes del proyecto.
- Desarrollo de una arquitectura al principio del proceso con el fin de minimizar riesgos y planificar el desarrollo.
- Evolucionar para obtener una retroalimentación continua en aras de mejorar.

Los proyectos de desarrollo que sean dirigidos por la metodología Open Up constarán de 4 fases: Inicio, Elaboración, Construcción y Transición, las cuales constan de iteraciones y estas orientadas por un mismo objetivo: la construcción de un software funcional y con calidad.

1.15 Lenguajes de modelado

Después de haberse seleccionado la metodología a emplear y seguir la guía que esta ofrece para el desarrollo de un software se hace necesario llevar a diferentes tipos de diagramas las acciones y necesidades que se detectan para una mejor comprensión de las mismas. Lo anterior lleva al uso de un lenguaje de modelado para esquematizar estas funciones con el objetivo de que sean mejor entendidas por todo el personal involucrado en el en proyecto.

1.16 UML

UML1[1] es un lenguaje de modelado visual que se emplea para graficar, generar y documentar artefactos en un sistema de software. Contiene un conjunto de notaciones específicas para la programación orientada a objetos las cuales se combinan con simbologías para lograr los distintos diagramas que representan las etapas del desarrollo de software en un proyecto.

UML se conforma de varios tipos de diagramas de forma estándar que le permite al analista crear un anteproyecto de varias facetas que sean comprensibles para los clientes, desarrolladores y todo los que se encuentran inmersos en el proceso de desarrollo. Todos esos diagramas son necesarios puesto que cada uno se dirige a una persona con un rol específico en el sistema. Un modelo UML indica qué es lo que debe hacerse pero no explica el cómo**.** (11)

Ventajas de UML

- Es un lenguaje de modelado de propósito general que pueden usar todos los modeladores, no es un método de desarrollo. No explica cómo pasar del análisis al diseño, ni

[1]UML: Lenguaje de Modelado Unificado.

del diseño a la implementación.

- Es independiente del ciclo de desarrollo que se vaya a seguir, soporta metodologías de desarrollo de software, pero no especifica qué metodología usar.

- No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática.

Pretende abordar los problemas actuales del desarrollo del software y en cualquier plataforma tecnológica donde se utilizará.

1.17 Conclusiones del capítulo

En este capítulo se describe la fundamentación teórica de la investigación realizada para el posterior análisis, diseño e implementación de la solución. Se exponen además la metodología de desarrollo, el lenguaje de modelado y la herramienta que se utilizará para ello. Se estableció como lenguaje de programación Python, como biblioteca de componentes gráficos a Gtk3 y como aplicaciones a usar en el entorno de desarrollo a Geany y Glade.

Capítulo 2: Diseño de la propuesta de solución

Introducción

En este capítulo se muestran los distintos componentes en el proceso de diseño de la aplicación. La implementación de estos componentes garantizará la correcta elaboración de la aplicación.

2.1 Descripción de la propuesta de solución

Para alcanzar el completo diseño de la solución se procede a desarrollar el proceso ingenieril en la creación de una aplicación. Se definen el(los) actor(es) que acceden al sistema, se levantan los requisitos funcionales para describir las posibles acciones a realizar en la aplicación y que serán iniciados por el actor en forma de casos de uso y los no funcionales para establecer las prestaciones mínimas para la correcta ejecución de la aplicación. Se diseñan el diagrama de clases, de vista lógica y de procesos del negocio. Luego de diseñar los diagramas se concluye enunciando los patrones a utilizar y se aplican al sistema en diseño.

Luego de realizado el diseño el software contará con dos escenarios principales que son:

-Iniciar, o detener los estados de ejecución de los servicios.

-Gestionar los niveles de ejecución de los mismos.

2.2 Actor del sistema

Un actor del sistema es una entidad que inicia una funcionalidad o brinda información al mismo. (19)

Tabla 1. Actor del sistema

Nombre de Actor	Descripción
Usuario	El usuario es en este caso una persona que interactuará con el sistema para gestionar los estados o los niveles de ejecución de los servicios.

2.3 Diagrama de procesos del negocio

Los diagramas de procesos del negocio forman parte del modelado del mismo. Al realizar este diagrama podemos ver de forma clara todas las actividades y su flujo lógico. Esta debe ser la forma de pensamiento lógico cuando diseñamos un software. (19)

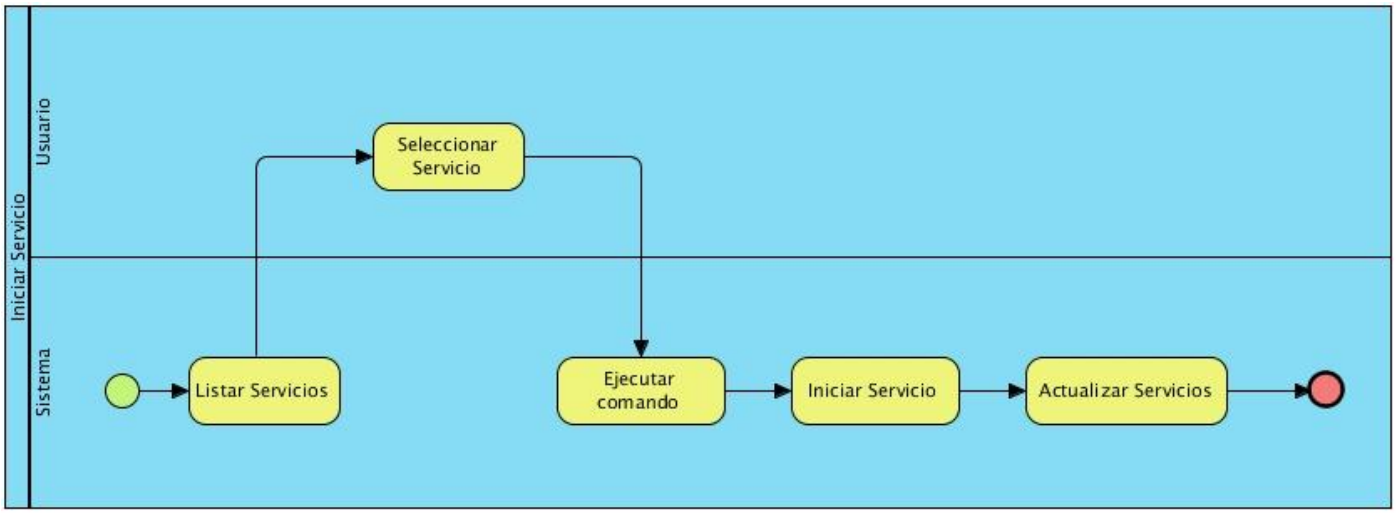

Figura 1. Diagrama de proceso Iniciar Servicio

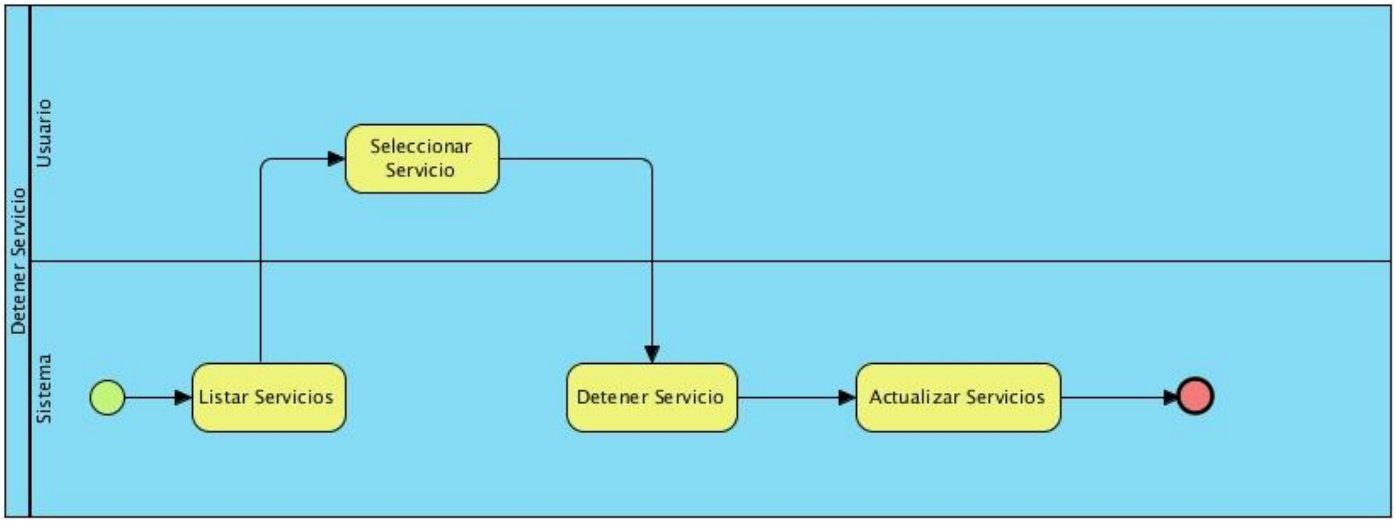

Figura 2. Diagrama de proceso Detener Servicio

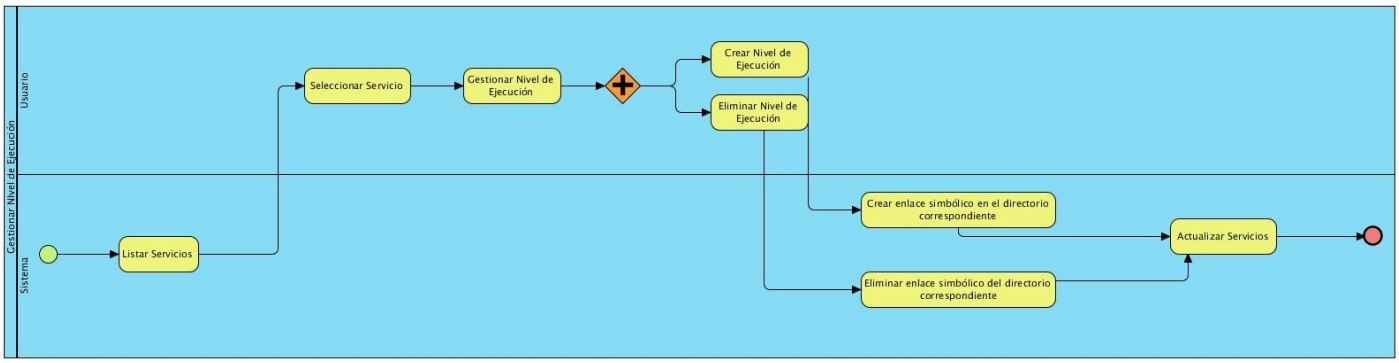

Figura 3 Diagrama de proceso Gestionar Nivel de Ejecución

2.4 Requisitos funcionales

Definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. (20)

CU 1. RF1. Iniciar Servicio.

CU 2. RF2. Detener Servicio.

CU 3. Gestionar nivel de Ejecución del Servicio.

– RF3. Crear nivel de Ejecución del Servicio.

– RF4. Eliminar nivel de Ejecución del Servicio.

2.5 Requisitos no funcionales

Tienen que ver con las características que de una u otra forma puedan limitar al sistema. Describen una restricción sobre el sistema que limita nuestra elección en la construcción de una solución. (20)

Interfaz.

– RNF 1. El software deberá poseer una interfaz gráfica de usuario que le proporcione interacción a los usuarios con la gestión de los servicios.

Software.

– RNF 2. La aplicación podrá ser utilizada en cualquier distribución similar a Nova 4.0 y que posea la biblioteca de Gtk3 y el intérprete de python 2.7.

Confidencialidad.

- RNF 3. Las acciones que requieran interacción con los servicios se ejecutarán bajo los privilegios establecidos por la aplicación.

2.6 Diagrama de Casos de Uso del Sistema

Un diagrama de Casos de uso captura dos partes fundamentales, lo que está afuera del sistema (actores) y lo que debe realizar el sistema (casos de uso). (21)

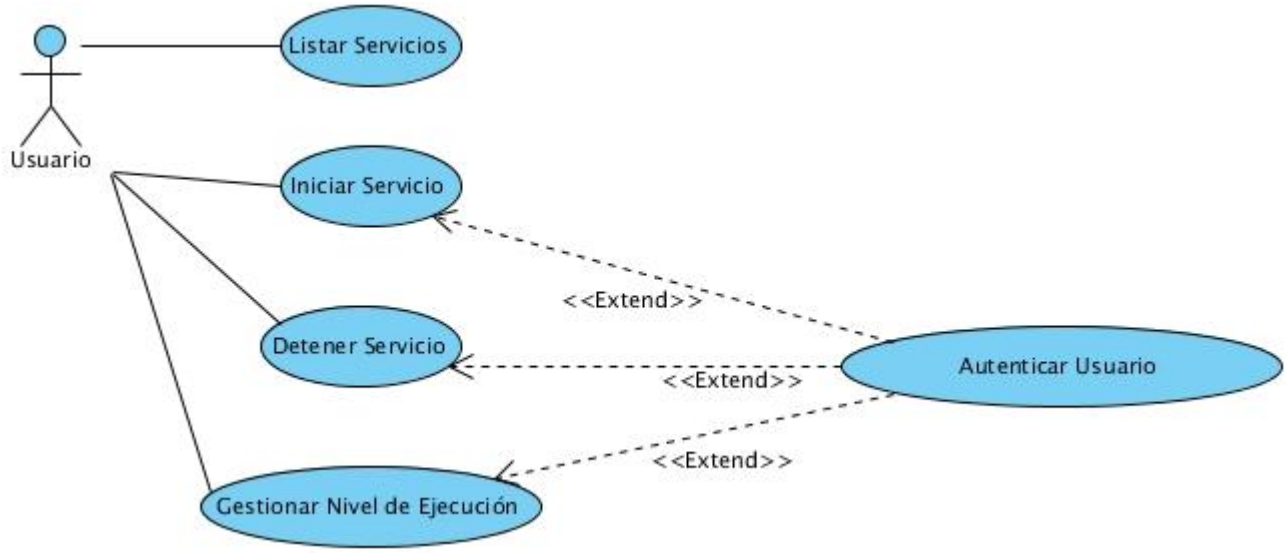

Figura 4. Diagrama de Casos de Uso del Sistema

2.7 Descripción de los Casos de Uso del Sistema

Los requisitos funcionales y el Diagrama de Casos de Uso del Sistema anteceden a la descripción de las funcionalidades que puede ofrecer el sistema.

Tabla 2. Caso de Uso Iniciar Servicio

Caso de Uso:	Iniciar Servicio
Tipo:	Funcional
Actores:	Usuario
Resumen:	El CU inicia cuando el usuario necesita iniciar algún servicio.
Precondiciones:	
Referencia(s):	RF1
Prioridad:	Alta
Complejidad:	Normal
Flujo normal de eventos	
Acción del actor	Respuesta del sistema
	1. El sistema muestra una interfaz con los servicios disponibles y sus estados de ejecución.
2. El usuario activa por medio de un	3. El sistema inicia el servicio indicado.

componente el servicio que quiere iniciar.	
Flujos alternos	
Acción del actor	Respuesta del sistema
	3.1. Si el usuario no se ha autenticado el sistema muestra un agente de autenticación.
4. El usuario se autentica en el sistema.	5. Si la contraseña no es correcta el agente de autenticación le notifica al usuario.
	3.2. El sistema notifica de que no se pudo iniciar el servicio.

Tabla 3. Caso de Uso Detener Servicio

Caso de Uso:	Detener Servicio
Tipo:	Funcional
Actores:	Mantenedor
Resumen:	El CU inicia cuando el usuario necesita detener un servicio.
Precondiciones:	
Referencia(s):	RF2
Prioridad:	Alta
Complejidad:	Normal
Flujo normal de eventos	
Acción del actor	Respuesta del sistema
	1. El sistema muestra una interfaz con los servicios disponibles y sus estados de ejecución.
2. El usuario activa por medio de un componente el servicio que quiere detener.	3. El sistema detiene el servicio indicado.

Flujos alternos	
Acción del actor	Respuesta del sistema
	3.1. Si el usuario no se ha autenticado el sistema muestra un agente de autenticación.
4. El usuario se autentica en el sistema.	5. Si la contraseña no es correcta el agente de autenticación le notifica al usuario.
	3.2. El sistema notifica de que no se pudo detener el servicio.

Tabla 4. Gestionar nivel de Ejecución del Servicio

Caso de Uso:	Gestionar nivel de Ejecución del Servicio
Tipo:	Funcional
Actores:	Mantenedor
Resumen:	El CU inicia cuando el usuario necesita gestionar el nivel de ejecución de un servicio.
Precondiciones:	
Referencia(s):	RF3, RF4
Prioridad:	Alta
Complejidad:	Normal
Flujo normal de eventos	
Acción del actor	Respuesta del sistema
	1. El sistema muestra una interfaz con los servicios disponibles y sus niveles de ejecución.
2. El usuario lleva a cabo una de las siguientes acciones: -Crear nivel de ejecución.	

-Eliminar nivel de ejecución.	
Escenario Crear Nivel de Ejecución	
Acción del actor	Respuesta del sistema
1. El usuario selecciona el nivel de ejecución que desea crearle a un servicio en específico.	2. El sistema crea el nivel de ejecución indicado.
Escenario Crear Nivel de Ejecución	
Acción del actor	Respuesta del sistema
1. El usuario selecciona el nivel de ejecución que desea eliminarle a un servicio en específico.	2. El sistema elimina el nivel de ejecución indicado.
Flujos Alternos: Escenario Crear Nivel de Ejecución	
Acción del actor	Respuesta del sistema
	2.1. Si el usuario no se ha autenticado en el sistema se muestra un agente de autenticación.
3. El usuario se autentica en el sistema.	
4. El usuario entra un nivel en el que el servicio se detiene.	5. El sistema notifica que no se puede iniciar y detener un servicio en el mismo nivel de ejecución.
Flujos Alternos: Escenario Eliminar Nivel de Ejecución	
Acción del actor	Respuesta del sistema
	2.1. Si el usuario no se ha autenticado en el sistema se muestra un agente de autenticación.
3. El usuario se autentica en el sistema.	
4. El usuario entra un nivel en el que el servicio se inicia.	5. El sistema notifica que no se puede detener e iniciar un servicio en el mismo nivel de ejecución.

2.8 Diagrama de Clases del Diseño

Un diagrama de clases representa la estructura de las clases del diseño de la solución en cuestión. Este diagrama se compone de los nombres de las clases, sus atributos, funcionalidades, relaciones y la cardinalidad que tomarán estas relaciones. (19)

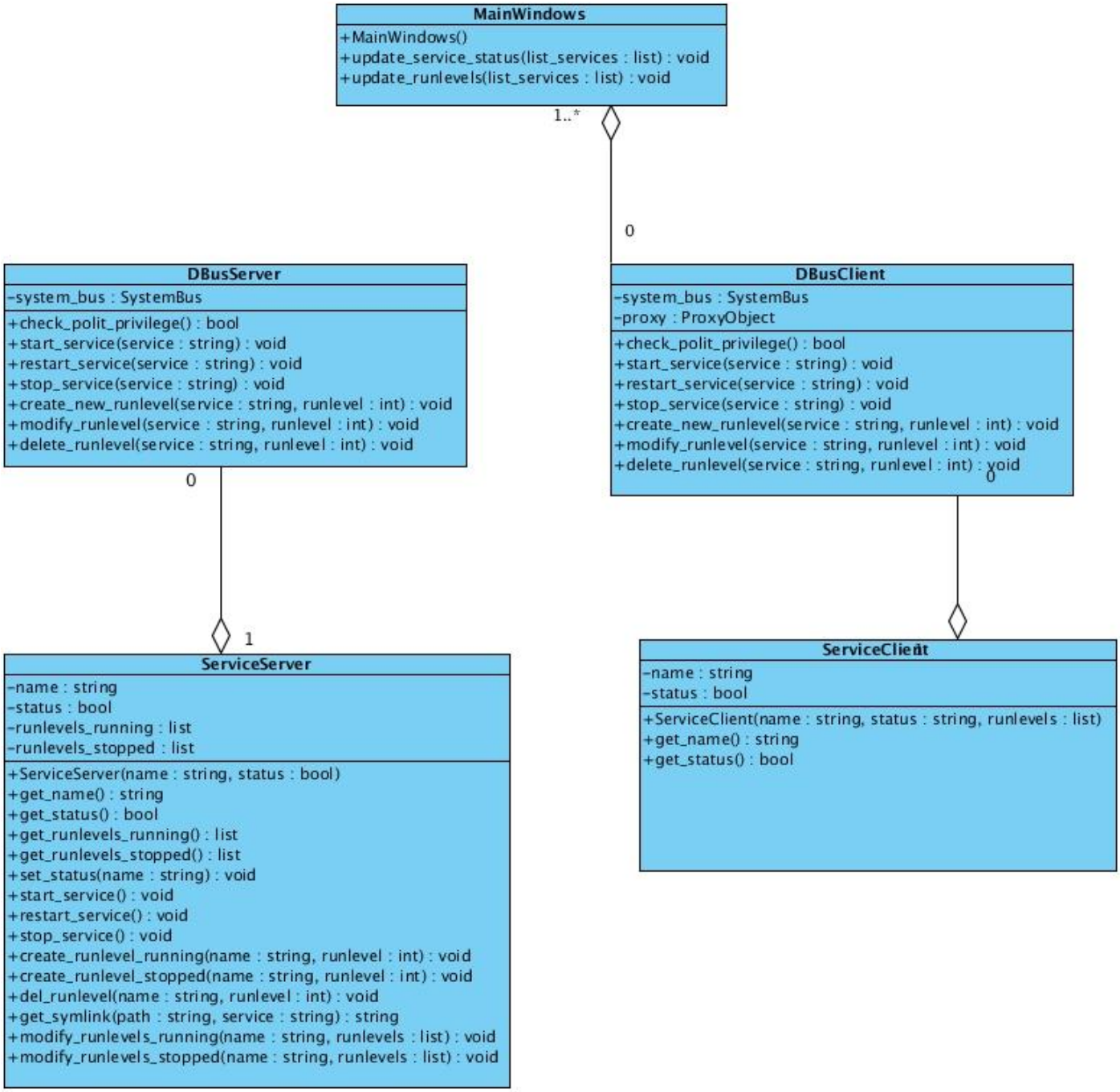

Figura 5. Diagrama de Clases del Diseño

2.9 Diagrama de Vista Lógica

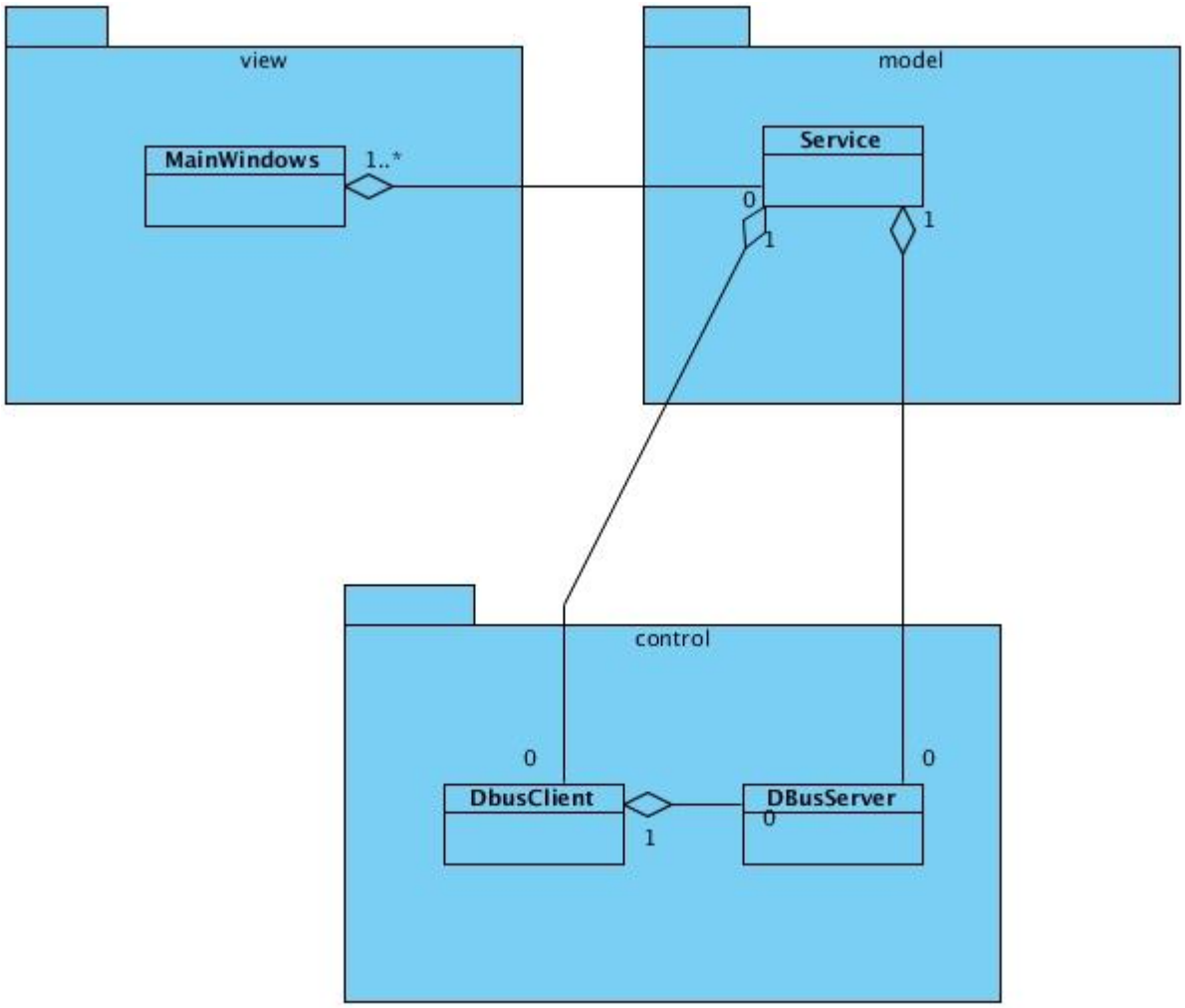

Figura 6. Diagrama de Vista Lógica

2.10 Patrones de Diseño

Una arquitectura orientada a objetos bien estructurada contiene diversos patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones tornan las arquitecturas más pequeñas, simples y entendibles. (22)

2.10.1 Patrones Grasp

Los patrones GRASP (General Resposibility Assigment Software Patterns por sus siglas en inglés) describen los principios fundamentales del diseño de objetos atendiendo a la asignación de responsabilidades. Constituyen la base del entendimiento para la creación de objetos especiales y aplica el razonamiento para el diseño del sistema de una forma más explicable. (22)

Sobresalen en GRASP 5 patrones fundamentales:

- Experto
- Creador
- Alta Cohesión
- Bajo acoplamiento
- Controlador

En el presente trabajo se usarán los siguientes patrones:

Experto: Asignar una responsabilidad al experto en información, la clase que tiene la información necesaria para la realización de la asignación. En el sistema la clase experta es la clase Servicio, que es la que contiene la información acerca de los servicios del sistema operativo.

Creador: Asignar a la clase B la responsabilidad de crear una instancia de la clase A si se cumple uno o más de los siguientes casos:

- B agrega objetos a A
- B contiene objetos de A
- B registra instancias de objetos de A
- B tiene datos de inicialización que se pasarán a un objeto A cuando sea creado (por tanto, B es un experto con respecto a la creación de A)
- B es un creador de los objetos A.

En el sistema son creadoras las clases DbusServer y DbusClient que contienen objetos de la clase Service y la clase MainWindows que contiene objetos de la clase DBusClient.

Alta Cohesión: Una clase con alta cohesión mejora la claridad y facilidad de uso, su mantenimiento se simplificará y es fácil de reutilizar. Todas las clases del sistema poseen alta cohesión, su contenido es bastante simple y permisible de reutilización.

Bajo Acoplamiento: Un elemento con bajo acoplamiento no depende demasiado de otros elementos. Una clase con alto acoplamiento confía en muchas otras clases, tales clases podrían no ser deseables por estar condicionadas a un complejo entendimiento, son difíciles de utilizar y los cambios realizados en las clases relacionadas conllevan a cambios locales. En el sistema, a pesar de no haber muchas clases las clases que dependen de otras solamente dependen de una sola clase.

2.11 Conclusiones del capítulo

En este capítulo se abordó el diseño de la aplicación por medio de la metodología Open Up y el lenguaje de modelado UML. Se definieron los requisitos funcionales y no funcionales, se realizó el diagrama de Casos de Uso del Sistema, diagrama de Clases del Diseño, diagrama de Vista Lógica, diagramas de Procesos y se aplicaron los patrones GRASP.

Capítulo 3. Implementación de la aplicación

Introducción

La implementación de una aplicación comienza cuando ha concluido el diseño de la misma. Se realiza un último diseño para el diagrama de componentes el cual definirá la arquitectura de los nodos físicos que conformarán la aplicación.

3.1 Diagrama de componentes

El diagrama de componentes muestra los elementos que se usarán en el sistema. UML define cinco estereotipos estándar que se le pueden aplicar a cada componente, estos son: ejecutables, bibliotecas, tabla, archivo y documento, los componentes se pueden agrupar en paquetes siguiendo un criterio lógico con vista a simplificar la implementación. (19)

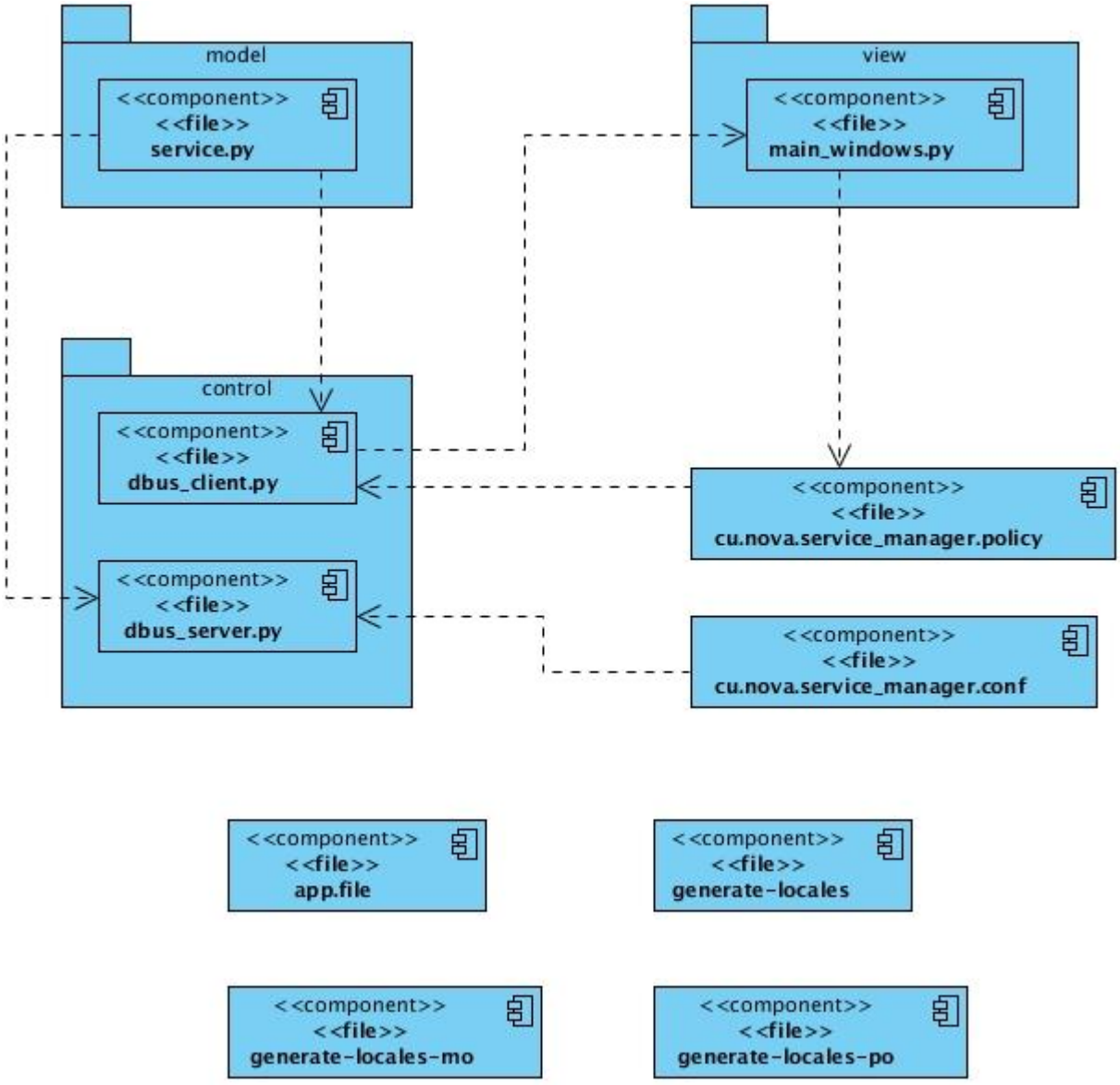

Figura 7. Diagrama de Componentes

3.2 Internacionalización

Con el objetivo de ofrecer la aplicación en varios idiomas se le hace un proceso de internacionalización. En este caso se le incluyeron los idiomas Inglés y Español utilizando el módulo gettext de python.

3.3 Empaquetamiento

En el proyecto de Nova Escritorio se realiza el desarrollo de varias aplicaciones de escritorios con el objetivo de enriquecer el entorno de escritorio de Nova. Cada vez que se termina la implementación y prueba de una aplicación esta se empaqueta con la intención de subirla a los repositorios de esta distribución cubana de GNU/Linux para para brindar la posibilidad de que sea más accesible de instalar por parte de los usuarios, así como también que sea más fácil de distribuir. Todo el proceso de empaquetamiento que se realiza en Nova está basado en la guía de mantenedores de Debian (23) la cual establece un conjunto de procesos a utilizar por medio de varias aplicaciones para lograr un correcto empaquetamiento.

3.4 Pruebas

Antes de liberar un producto de software se le hacen unas pruebas al mismo con el objetivo de abarcar el mayor margen de errores posibles en el sistema. Una prueba es una actividad en un sistema o componente ejecutado bajo condiciones específicas para ser observados y corregir futuros errores. Las pruebas de software son elementos claves para lograr la calidad esperada por el usuario final, se encargan de revisar especificaciones en el diseño y en el código de programación.

3.4.1 Pruebas de Caja Negra

Las pruebas de caja negra o pruebas de comportamiento se enfocan en los requisitos funcionales del sistema e intentan encontrar errores tales como: funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos, errores de rendimiento y errores de inicialización o terminación. (21)

Entre los diversos métodos de prueba de caja negra uno de los más usados es la partición equivalente: este método divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal descubre de forma inmediata una clase de errores que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar.

Tabla 5. Escenario: Estados de ejecución.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.1 Iniciar servicio.	Iniciar un servicio indicado.	Si el usuario está autenticado bajo los	1. Activar servicio por medio del componente.

		privilegios que requiere la aplicación se inicia el servicio.	
EC 1.2 Iniciar servicio.		Si el usuario no está autenticado bajos los privilegios que requiere la aplicación se muestra un agente de autenticación de polkit.	
EC 1.3 Iniciar servicio.		El sistema ofrece un mensaje de que no se pudo iniciar el servicio.	
EC 1.4 Detener servicio.	Detener un servicio indicado.	Si el usuario está autenticado bajo los privilegios que requiere la aplicación se detiene el servicio.	1. Desactivar servicio por medio del componente.
EC 1.5 Detener servicio.		Si el usuario no está autenticado bajos los privilegios que requiere la aplicación se muestra un agente de autenticación de polkit.	
EC 1.6 Detener servicio.		El sistema ofrece un mensaje de que no se pudo detener el servicio.	

Tabla 6. Escenario: Niveles de ejecución.

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 2.1 Crear niveles de ejecución como iniciado.	Añadirle un nivel de ejecución al servicio.	Si el usuario está autenticado bajo los privilegios que requiere la aplicación se le añade el nivel al servicio como iniciado.	1. Seleccionar el nivel de ejecución que se quiere adicionar como iniciado.
EC 2.2 Crear niveles de ejecución como iniciado.		Si el usuario no está autenticado bajos los privilegios que requiere la aplicación se muestra un agente de autenticación de polkit.	
EC 2.3 Crear niveles de ejecución como iniciado.		Si el servicio se detiene en ese nivel de ejecución el sistema muestra un mensaje de que no puede haber concurrencia de iniciado y dentenido.	
EC 2.4 Detener servicio.	Detener un servicio indicado.	Si el usuario está autenticado bajo los privilegios que requiere la aplicación se le añade el nivel al servicio como detenido.	1. Seleccionar el nivel de ejecución que se quiere adicionar como detenido.
EC 2.5 Detener servicio.		Si el usuario no está autenticado bajos los privilegios que requiere la aplicación se muestra un agente de autenticación	

		de polkit.	
EC 2.6 Detener servicio.		Si el servicio se inicia en ese nivel de ejecución el sistema muestra un mensaje de que no puede haber concurrencia de iniciado y dentenido.	

3.5 Conclusiones

En este capítulo se llevó a cabo la implementación de la aplicación a partir de los componentes expuestos en el epígrafe 3.1, se le incluyó la condición de internacionalización y posteriormente se le aplicó el proceso de empaquetamiento con el objetivo de incluir dicha aplicación en los repositorios de Nova 2013.

3.6 Conclusiones generales

A lo largo del documento actual se fue describiendo todo el proceso de análisis, diseño e implementación de una aplicación con interfaz gráfica de usuario que mejoró la interacción por parte de los usuarios con la gestión de los servicios en Nova 4.0. A dicha aplicación se le realizaron pruebas de caja negra con el objetivo de evaluar desde la interfaz que todos los requisitos planteados se cumplieran correctamente.

3.7 Recomendaciones

Se recomienda a los desarrolladores del proyecto Nova Base que mejoren las descripciones en los scripts de servicios con que cuenta la distribución cubana de software libre GNU/Linux Nova 4.0.

Glosario de términos

Multimedia: hace referencia a la aparición de imágenes, botones y cajas de entrada de texto.

Entornos de escritorio: es un entorno que brinda conjunto de aplicaciones al usuario que permiten usar la computadora de modo gráfico. Existen varios entornos de escritorio en la actualidad como el caso de GNOME, KDE, UNITY, XFCE, LXDE entre otros.

Distribución: variante del Sistema operativo GNU/Linux.

Kernel: núcleo de código del sistema operativo.

Bindings: enlaces del lenguaje para utilizar las bondades de la biblioteca.

Widgets: componentes multimedia especializados como barras de desplazamiento, switches (cambio de dos estados) entre otros.

Referencias Bibliográficas

1. **Marrero Exposito, Carlos.** *Interfaz gráfica de usuario. Diseño gráfico y comunicación visual.* Tenerife : s.n., 2006.

2. **The Linux Information Project.** Daemon Definition. [En línea] [Citado el: 15 de febrero de 2013.] http://www.linfo.org/daemon.html.

3. **González Duran, Sergio.** Uso y control de servicios en Linux. [En línea] [Citado el: 14 de enero de 2013.] http://www.linuxtotal.com.mx/index.php?cont=info_admon_003.

4. **The Linux Tutorial.** Linux Knowledge Base and Tutorial. [En línea] [Citado el: 17 de febrero de 2013.] http://www.linux-tutorial.info/modules.php?name=MContent&pageid=65.

5. [En línea] [Citado el: 20 de febrero de 2013.] http://www.upstart.ubuntu.com.

6. **Ubuntu Manpage.** service - executes System-V style init script actions. [En línea] [Citado el: 25 de mayo de 2013.] http://manpages.ubuntu.com/manpages/hardy/man8/service.8.html.

7. —. Debian Runlevel configuration tool. [En línea] [Citado el: 25 de mayo de 2013.] http://manpages.ubuntu.com/manpages/lucid/man8/rcconf.8.html.

8. **Oppegaard, Joe.** [En línea] [Citado el: 24 de mayo de 2013.] www: http://sysv-rc-conf.sourceforge.net.

9. Boot-Up Manager (docs). [En línea] [Citado el: 12 de febrero de 2013.] http://www.marzocca.net/linux/bumdocs.html#how.

10. **CNET.** GTK+ Insecure Library Loading Vulnerability - CNET Spyware, viruses, & security Forums. [En línea] [Citado el: 24 de mayo de 2013.] http://forums.cnet.com/7726-6132_102-5198162.html.

11. Visual Paradigm for UML. [En línea] [Citado el: 15 de febrero de 2013.] http://www.visual-paradigm.com.

12. [En línea] [Citado el: 19 de febrero de 2013.] http://www.python.org.

13. What is GTK+, and how can I use it. [En línea] [Citado el: 22 de febrero de 2013.] http://www.gtk.org/.

14. [En línea] [Citado el: 22 de febrero de 2013.] http://www.geany.org.

15. [En línea] [Citado el: 23 de febrero de 2013.] http://glade.gnome.org/.

16. [En línea] [Citado el: 1 de marzo de 2013.] http://www.freedesktop.org/wiki/Software/polkit.

17. [En línea] [Citado el: 1 de marzo de 2013.] http://www.freedesktop.org/wiki/Software/dbus.

18. [En línea] [Citado el: 4 de marzo de 2013.] http://epf.eclipse.org/wikis/openup/index.htm.

19. Elementos de UML. [En línea] [Citado el: 24 de mayo de 2013.] http://docs.kde.org/stable/es/kdesdk/umbrello/uml-elements.html.

20. Requerimientos funcionales y no funcionales. [En línea] [Citado el: 24 de mayo de 2013.] http://www.slideshare.net/Lismirabal/requerimientos-funcionales-y-no-funcionales.

21. **PRESSMAN, ROGER S.** *Ingienería de Software. Un enfoque práctico.* s.l. : McGraw-Hill, 2005. 84-481-3214-9.

22. **Saavedra, Jorge.** PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad).Parte II | El Mundo Informático. [En línea] [Citado el: 5 de abril de 2013.] http://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-patrones-de-software-para-la-asignacion-general-de-responsabilidadparte-ii/.

23. **Rodin, Josip, Aoki, Osamu y Hertzog, Raphael.** Debian New Maintainers' Guide. [En línea] [Citado el: 27 de mayo de 2013.] http://www.debian.org/doc/maint-guide/.