Universidad de las Ciencias Informáticas

Herramienta gráfica para efectuar la gestión de paquetes en la distribución de GNU/Linux Nova.

Trabajo de Diploma

Autores:

Angel Goñi Oramas

Mijail Hurtado Fedoróvich

Tutor:

Msc. Héctor Rodríguez Figueredo

Presentada en opción al Título de Ingeniero en Ciencias Informáticas

Ciudad Habana, Cuba

Junio, 2008

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 firmo la presente a los ____ días del mes de _____ del año _____.

Mijail Hurtado Fedoróvich	Angel Goñi Oramas	Héctor Rodríguez Figueredo
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

Agradecimientos

De Angel:

A Mijail, por compartir todo éste trabajo, y el que falta.

A Héctor y Delly, por el apoyo, la ayuda y la tutoría en estos 5 años, y por dejarme molestar en su casa después de las 9:30 pm.

A Daniel por los varios miles de líneas de código que ha escrito implementando Summon, y la paciencia que ha tenido conmigo al hacerlo.

A Bizerta, por hacernos cambiar media tesis, que conste que no es una crítica.

A mi mamá, porque también fue su tesis.

A Raidel, Félix, Abel, Migue y Román por convertir cada día, noche y madrugada de trabajo en algo divertido.

A Adis, Yuliette, Baby, Ana Maria y Mariannis, por existir, ir al laboratorio, y dejarnos hablar mal de las mujeres...

A la Revolución, por la oportunidad.

A todas aquellas personas que me han apoyado y ayudado durante todo éste tiempo, no alcanza una tesis para mencionarlos uno por uno, pero sigo estando eternamente agradecido.

De Mijail:

A Ángel por su batallar constante por nosotros.

A esta revolución y a mi Comandante Fidel Castro Rúz.

A Héctor, por su incondicional ayuda.

A mamá, por hacer que no me falte absolutamente nada.

A mi papá, sin tí todo hubiera sido muy diferente.

A mis amigos, no son muchos, pero hacen por miles, infinitas gracias por su apoyo.

Agradecimiento especial a mi novia Yenisleys por haber mantenido mi espíritu en alto a pesar de la distancia y los problemas.

Agradezco a Daniel por habernos ayudado muchísimo en la confección del software de nuestra tesis y a los demás integrantes del proyecto NOVA por apoyarnos con ideas y consejos.

A Roman, Abel, Miguel, Anamaría, Alberto, Mariannis, Felix, Raidel, por poder contar con ellos como un equipo para almorzar y comer todos los días.

Agradecer la ayuda de Mayra Oramas en todo momento asesorando la tesis.

Dedicatoria

De Angel:

A mi mamá, sobran las palabras.

A mi abuela, todavía hay mucha guerra que dar en el convento.

A mi abuelo, que éste sea un motivo más para sentir orgullo.

A los mingos, por los 5 años en las buenas y las malas.

A Yoandy y Anielkis, por enseñarme todo lo que se, e incluso lo que no he podido retener en mi cabeza.

A ExInferno, porque el auto halago y la auto ofrenda se valen, y por todo el ruido que hemos hecho...

Al equipo de desarrollo de Nova, porque si va.

De Mijail:

A mi mamá, por hacer de mi lo que soy hoy.

A mi papá, por sus sobrados consejos y su experiencia.

A mi abuela, por existir.

A mi tío Rubén, por ayudarme a soñar.

A los integrantes de Nova, que hacen que cada vez la estrella brille más.

A Yenisleys (mi novia), por ser mi impulso espiritual.

Resumen

Se propone un sistema informático que garantiza una mayor eficiencia y control sobre los procesos de instalación, desinstalación y actualización de software en el sistema operativo Nova; acciones que actualmente se realizan de forma ineficiente y engorrosa.

Se documenta la investigación realizada con el fin de demostrar que dada la situación problémica actual se requiere la creación de un nuevo software que responda a las necesidades encontradas.

Se realiza un estudio comparativo sobre las tecnologías y tendencias de los sistemas de gestión de paquetes de software, seleccionando las más adecuadas como guía para la elaboración de la solución propuesta.

Se propone la metodología ágil de desarrollo Programación Extrema, recogiéndose en el presente documento los principales artefactos generados en cada una de sus iteraciones.

Palabras clave

Gestor de paquetes, instalar, desinstalar, actualizar, dependencia, paquete precompilado, archivo binario.

Aclaración necesaria

Debido al carácter técnico del documento, durante la lectura del mismo, el lector puede hallar determinadas frases o palabras señaladas con cursiva, lo cual significa que las mismas se encuentran definidas en el glosario de términos para un uso más cómodo del documento.

El primer capítulo representa un recorrido de las técnicas, soluciones y metodologías existentes y que podrían relacionarse con los fines que se persiguen, aunque ninguna de estas se ajusta completamente a la solución del problema presentado.

El capítulo 2 presenta la solución del problema que se pretende resolver, contemplando la fase de exploración de los requerimientos del sistema a desarrollar.

El capítulo 3 contempla el proceso de análisis y diseño del software propuesto, describiéndose detalladamente mediante los artefactos generados por la metodología de desarrollo.

Finalmente, se hacen algunas recomendaciones y la evaluación de futuras líneas de investigación, se ofrecen los anexos complementarios así como la recopilación de la bibliografía relacionada y un glosario de términos que sirve de apoyo en la lectura de un documento tan técnico.

Índice de contenidos

Introducción

Antecedentes

Cuba se ha caracterizado siempre por ser un país consumidor de tecnologías informáticas, cuyos productores son, en la mayoría de los casos, transnacionales pertenecientes a países del primer mundo. Esta relación de consumo ha generado una dependencia tecnológica que ahoga el desarrollo de la naciente Industria de Software Cubana y aumenta la amplia brecha digital existente entre Latinoamérica y Europa y la América Anglosajona.[1]

Los sistemas operativos más utilizados en Cuba son de la familia *Microsoft Windows* los cuáles en la mayor parte de los casos son copias ilegales, pues por las leyes del Gobierno norteamericano, Microsoft no puede vender ni contratar servicios a un país “embargado”.

La situación antes descrita trae aparejada la imposibilidad de crear empresas que provean los servicios de forma nacional debido a que sería necesaria la aprobación de los “dueños” de las tecnologías sobre las que se va a trabajar. Otro gran problema es la atadura a los cambios de licencias y políticas en los sistemas extranjeros que afectarían toda la infraestructura nacional. Un ejemplo es el hecho de que desde el rango de *IP* perteneciente a Cuba no se pueda descargar ninguna actualización de software de servidores pertenecientes a *Sun Microsystems* o Microsoft Corporation.

La Universidad de las Ciencias Informáticas (UCI) y la sociedad cubana en general se enfrentan a un inminente proceso de migración a tecnologías de software libre con el objetivo de alcanzar un estatus de independencia y soberanía tecnológica de la nación cubana. Uno de los primeros pasos para asimilar el cambio que se avecina fue el desarrollo de un *sistema operativo* cubano basado en *códigos fuentes* distribuido bajo *licencias libres* que hiciera la función de plataforma sobre la cual se base el proceso de transición.

Nova, sistema operativo desarrollado en la UCI, surge como una alternativa competitiva a considerar como plataforma base para realizar el proceso de migración, caracterizándose por su enfoque centrado en la realidad de cubana, tal como plantean sus desarrolladores:

“En estos momentos el Proyecto Nova tiene como objetivo la creación de un sistema operativo, no la mera personalización de una distribución. Se aspira a proveer un producto orientado a usuarios

inexpertos que hayan tenido que migrar de Microsoft Windows a entornos *GNU/Linux* o cuya experiencia con computadoras sea nula. Se pretende automatizar la mayor cantidad de procesos posible de forma que la interacción de la persona con el sistema sea fácil e intuitiva facilitando el proceso de transferencia de conocimientos y aprendizaje, algo tan difícil cuando se trata de asimilar nuevas tecnologías.” [2]

Situación problémica.

Nova es un sistema operativo de la familia GNU/Linux pero adolece de algunas funcionalidades básicas cruciales para una total aceptación por los usuarios. De estas insuficiencias, la más importante es la carencia de una forma sencilla, cómoda e intuitiva de instalar y configurar nuevos programas en el sistema.

Actualmente la gestión de paquetes de software en Nova tiene las siguientes características:

- Construir el software utilizando los códigos fuentes es extremadamente lento.
- Requiere un conocimiento avanzado del sistema.
- El usuario corre el riesgo de romper aplicaciones e incompatibilizar su funcionamiento.
- El *sistema de gestión de paquetes* actual no está diseñado para el uso de *paquetes precompilados*.
- Se crean problemas al tratar de interactuar con algunas dependencias de software.
- Se ejecuta desde la *terminal* del sistema con acceso de *súper usuario (root)*.
- No existe una herramienta con *interfaz gráfica (GUI)* adecuada que facilite el trabajo.
- Cuando se *desinstala* un programa no se eliminan las *dependencias huérfanas* que quedan en el sistema.
- El proceso de actualización es complejo y trabajoso.

De las dificultades anteriormente expuestas se deriva la necesidad de crear o adaptar un sistema informático que optimice el proceso de gestión de paquetes en el sistema operativo Nova, que utilice el método de gestión de *archivos binarios* y mejore el *rendimiento* en cuanto al *consumo de recursos* y la velocidad del proceso.

Problema científico

La carencia de una herramienta para efectuar la gestión de paquetes de software en la distribución de GNU/Linux Nova.

Objeto de estudio

El objeto de estudio de este trabajo es el proceso de gestión de paquetes de software en los sistemas operativos.

Campo de acción

El proceso de gestión de paquetes de software en la distribución NOVA

Idea a defender

La elaboración de una herramienta gráfica que implemente diferentes métodos de gestión de paquetes de software con núcleo Linux optimizará el proceso de gestión de paquetes de software en la distribución Nova.

Objetivo general

Desarrollar un sistema informático que permita la optimización del proceso de gestión de paquetes de software la distribución Nova.

Objetivos específicos

1. Analizar el proceso de instalación y desinstalación en los diferentes sistemas operativos haciendo énfasis en los que utilizan núcleo Linux.
2. Identificar los diferentes sistemas de gestión de paquetes de software disponibles en el mercado.

3. Valorar críticamente las tendencias de los lenguajes de programación, las herramientas técnicas y las metodologías utilizadas.
4. Realizar el proceso de ingeniería de software para la elaboración del sistema informático propuesto como solución.
5. Realizar un prototipo de interfaz para la herramienta gráfica propuesta.

Tareas de la investigación

Estudiar el proceso de *instalación* en los diferentes sistemas operativos, haciendo énfasis en los que poseen núcleo Linux.

1. Análisis de los procesos gestión de paquetes de software en los sistemas operativos.
2. Identificación de los sistemas de gestión de paquetes de software disponibles en el mercado.
3. Valoración crítica de las tendencias de los lenguajes de programación, herramientas y metodologías utilizadas.
4. Realización del proceso de ingeniería de software a la herramienta propuesta.
5. Elaboración de un prototipo de interfaz para la herramienta gráfica a desarrollar.
6. Validación de la herramienta gráfica propuesta.

Métodos de investigación

Para cumplir las tareas de la investigación se utilizan los siguientes métodos teóricos:

1. Método de modelación como herramienta para comprender el problema y crear un modelo abstracto del mismo que permita el diseño de la solución.
2. Método histórico-lógico para, enfocados en el objeto de estudio, analizar su desarrollo, evolución y posible aplicación en el problema.

1. Fundamentación del tema

1.1. Introducción

En el presente capítulo se describe la situación actual de la gestión de paquetes de software en los diferentes sistemas operativos, se describe el proceso de instalación y desinstalación de programas informáticos haciendo énfasis la detección de los problemas más comunes. Se hace un estudio comparativo de las herramientas de gestión de paquetes más relevantes del mundo y de las tecnologías utilizadas para desarrollarlos con el fin de adoptar las más adecuadas para la solución del problema.

El estudio se enfoca pensando en las características de Nova como distribución, con un *ciclo de liberación* rápido, que exige facilidad de mantenimiento del código fuente y agilidad en el proceso de desarrollo de las herramientas de software desarrolladas por el equipo del proyecto.

1.2. Proceso de instalación de software en los sistemas operativos.

La instalación de programas de software es el proceso por el cual nuevos programas son transferidos a una computadora y configurados, para ser usados con el fin para el cual fueron desarrollados.

Una instalación exitosa es una condición necesaria para el correcto funcionamiento de cualquier software. Mientras más complejo sea el software, más archivos contenga, mayor sea la dispersión de los archivos y la interdependencia con otro software, mayor es el riesgo de alguna falla durante la instalación. Si el proceso de instalación falla aunque sea sólo parcialmente, el fin perseguido no podrá ser alcanzado por lo que sobre todo en casos de programas complejos, el desarrollo de un proceso de instalación confiable y seguro es una parte fundamental del desarrollo.

En los últimos años se han desarrollado normas y técnicas cada vez más potentes para simplificar y estandarizar el proceso de instalación de software. Para el mismo se pueden aplicar las siguientes técnicas básicas:

Técnica 1. La copia de los archivos en algún lugar del directorio:

1.1. Este es un sistema fácil e intuitivo, preferido en *Mac OS*. Uno de sus riesgos es que las versiones más antiguas del mismo programa hayan quedado abandonadas en algún otro

lugar sin que el usuario lo note.

Técnica 2. La copia de un instalador, el que posteriormente procesa el la instalación software deseado.

Técnica 3. Existencia de algún servicio de gestión de software que se ocupa de instalar un paquete de software. (Sistema de gestión de paquetes).

1.2.1. Pasos a seguir para la instalación de software.

1. Verificación de la compatibilidad:

   Se debe comprobar si se cumplen los requisitos para la instalación en cuanto a hardware y software, a veces es necesario desinstalar versiones antiguas del mismo software.

2. Verificación de la integridad:

   Se verifica que el paquete de software es el original para evitar la instalación de *programas maliciosos*.

3. Creación de los directorios necesarios para el paquete:

   Para mantener el orden en el "*directorio Raíz*", cada sistema operativo tiene un estándar para la instalación en ciertos directorios, un ejemplo es la *Linux Standard Base*.

4. Creación de los usuarios:

   Existe una jerarquía de usuarios en el sistema capaces de administrar o usar determinadas aplicaciones.

5. Concesión de los permisos:

   Para controlar el sistema y limitar posibles daños, se les conceden a los usuarios el mínimo de permisos necesarios.

6. Copia, desempaque y descompresión de los archivos desde el paquete de software:

   Para ahorrar *ancho de banda* y tiempo en la transmisión por Internet o espacio de disco duro, los paquetes vienen empaquetados y comprimidos.

7. Compilación y enlaces con librerías:

   Suele pasar que en algunos casos no se actualizan los enlaces con algunas librerías, por

lo que se ve afectada la estabilidad o funcionamiento de algún software.

8. Configuración:

Este paso es de gran importancia, ya que del éxito del mismo depende la estabilidad o uso del software y del sistema. Aquí se practican configuraciones automáticas en el sistema y el software antes de ser instalado y después de instalado. El objetivo principal de este paso es adaptar el software a la distribución que lo use.

9. Definir las *variables de entorno* particulares del software y del sistema:

Este paso es inherente a la configuración pero en la mayoría de los casos suele ser decisivo para el buen funcionamiento del software. Algunos comportamientos del software solo pueden ser determinados por medio de estas variables, un proceso de la etapa de configuración más dinámico.

10. Adquisición de la licencia que registra y activa el software:

Para el software comercial a veces el desarrollador o compañía de software exige el registro de la licencia para instalación o activación del mismo, tal es el caso de la herramienta case “Visual Paradigm”.

1.2.2. Proceso de desinstalación de software en los sistemas operativos.

Para lograr una correcta desinstalación no solo deben ser borrados los archivos de la raíz, sino que también se deben deshacer cambios en otros aspectos como pueden ser eliminar usuarios que hayan sido creados por concepto de instalación del software, retirar consecuentemente permisos, borrar directorios, y también llevar un registro o *log* de los cambios realizados.

Debido a la creciente complejidad de los sistemas operativos y sus *interfaces (API)*, la desinstalación de software puede ser no solo contraproducente sino también poner en peligro la estabilidad del sistema. La calidad de un software no solo depende de sus efectos productivos o creativos sino también de su capacidad de integración en el sistema operativo y la compatibilidad con otros programas. El desarrollador del software debe ofrecer una función para desinstalar su software sin dañar o desestabilizar el sistema.

1.2.3. Problemas en la gestión de paquetes de software.

El proceso de gestión de paquetes de software en los sistemas operativos es sumamente complejo y riguroso, dicha complejidad depende de factores tales como: métodos utilizados para su gestión, complejidad que estos tengan y nivel de integración del software con otro software, biblioteca o sistema operativo en general.

El problema más común es el provocado por las bibliotecas compartidas por los diferentes programas y sus versiones en el momento de la instalación y desinstalación. En los sistemas operativos de la familia Windows, los programas están comprimidos en binarios autoextraíbles con todas sus dependencias lo que simplifica mucho el proceso pero provoca un fenómeno llamado infierno de las DLL que se abordará mas adelante.

En los sistemas de tipo UNIX y GNU/Linux los sistemas de paquetería solucionan ese problema gestionando automáticamente todas las necesidades de la instalación en el momento de realizar el proceso, descargan los programas requeridos de forma independiente y los procesan en el orden correcto. En caso de que sea necesaria la convivencia de más de una versión de una biblioteca en el sistema, se le asigna un identificador que evita el conflicto de funcionamiento entre ellas.

1.2.3.1. Infierno de las DLLs.

El término DLL Hell (infierno de las DLLs) se refiere a los problemas ocasionados por las DLLs en los sistemas operativos de la familia Microsoft Windows. En Windows estas bibliotecas están muy extendidas y son compartidas por múltiples aplicaciones (por ejemplo, la *MFC* está compartida por prácticamente todas las aplicaciones gráficas). De esta compartición de código surgen dos problemas que constituyen el "infierno":

Al instalar un programa se reemplaza una DLL por otra versión incompatible (conflicto de versiones) Al desinstalar un programa se borra una DLL compartida. En ambos casos los programas que compartan la DLL dejarán de funcionar con los consiguientes trastornos que supone.

Existen un conjunto de soluciones manuales para evitar estos problemas:

- Incorporar el número de versión a las DLLs para evitar sobrescribirlas con versiones incompatibles: Este sistema se usa en la MFC y podemos encontrar, por ejemplo, las bibliotecas MFC41.dll y MFC70.dll en el mismo sistema. El problema es que al final existen

varias versiones de las DLLs instaladas aunque en la actualidad no supone un gran trastorno debido a la capacidad de los discos.

- Fijar el comportamiento y no permitir modificaciones en el mismo, de esta forma se evitaría la existencia de DLLs no compatibles.

Las versiones más modernas de Windows proponen soluciones automáticas para el problema (mayor control del versionado, petición al usuario de qué DLL quiere mantener, etc.) pero las antiguas instalaciones siguen corrompiendo las DLLs. Por ello se han planteado dos soluciones:

- Scripts de instalación MSI:

  Se trata de pequeñas bases de datos que indican qué ficheros y versiones instala una aplicación. De esta forma es posible determinar qué versiones son compatibles y cuales no o volver a instalar versiones en el caso de instalaciones corruptas.

- Ensamblados:

  Es un concepto que aparece con *.NET* y consiste en un fragmento de código ejecutable (DLL o EXE) junto a un fichero (que puede estar incorporado como recurso en el fichero) que indica qué contiene, versiones, etc. De esta forma es posible instalar varias versiones diferentes del mismo ensamblado y cargar la que necesita cada programa usando esa información.

1.3. Wizards independientes y sistemas de gestión de paquetes.

En la gestión de paquetes de software existen dos grandes diferencias respecto a la filosofía que se maneja y la forma de procesar los paquetes a la hora de ser instalados:

1. instalador independiente con asistente o guía (en inglés wizard, comúnmente usados en Windows).
2. *sistema de gestión de paquetes* centralizado.

1.3.1. Diferencias.

Comparación entre instaladores y sistemas de gestión de paquetes.

Sistema de Gestión de Paquetes	Instalador
Forma parte del sistema operativo.	Cada producto viene unido a su propio instalador.
Usa una única base de datos de instalación.	Rastrea su propia instalación
Puede verificar y administrar todos los paquetes sobre el sistema.	Sólo trabaja con su propio producto.
Un único suministrador de sistema de administración de paquetes.	Múltiples vendedores de instalador.
Un único formato de paquetes.	Múltiples formatos de instalación

1.3.2. Similitudes.

Los sistemas de gestión de paquetes cumplen la función de organizar todos los paquetes instalados en el sistema y se encargan de mantener su usabilidad. Esto se consigue combinando las siguientes técnicas:

- Comprobación de la suma de verificación para evitar que haya diferencias entre la versión local de un paquete y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación simple de paquetes.
- Resolución de dependencias para garantizar que el software funcione correctamente.
- Búsqueda de actualizaciones para proveer la última versión de un paquete, ya que normalmente solucionan errores (en inglés bugs) y proporcionan actualizaciones de seguridad.
- Agrupamiento de paquetes según su función para evitar la confusión al instalarlos o mantenerlos.
- Muchos de los sistemas de gestión de paquetes ampliamente utilizados utilizan algún backend simple para instalar los paquetes. Por ejemplo, YUM utiliza RPM como backend y APT utiliza dpkg.

1.4. Familias de sistemas de gestión de paquetes.

La mayoría de los sistemas de gestión de paquetes de software están compuestos por varios programas independientes que actúan como las capas de una arquitectura. Debido a esa composición, el análisis de los principales gestores de paquetes se hará describiendo cada una de las herramientas que lo componen:

- Interfaz gráfica *(GUI)*.
- Interfaz de línea de comandos *(CLI)*.
- Librerías que proveen las funcionalidades requeridas.

1.4.1. Sistemas basados en paquetes binarios.

1.4.1.1. DPKG, APT e interfaces derivadas.

DPKG es la base del sistema de gestión de paquetes de Debian GNU/Linux y fue creado por Ian Jackson en 1993. Es una herramienta de bajo nivel que necesita una interfaz de alto nivel para realizar tareas de resolución de dependencias, descarga de binarios de Internet y otras tareas complejas, función que cumple muy acertadamente *APT*.

DPGK está compuesta por varías herramientas simples las cuales permiten realizar todo el trabajo de gestión de binarios:

- dpkg-source empaqueta y desempaqueta los archivos fuentes de un paquete Debian.
- dpkg-gencontrol lee la información de un árbol fuente Debian desempaquetado y genera un paquete binario de control, creando una entrada para éste en el fichero debian/files.
- dpkg-shlibdeps calcula las dependencias de ejecutables respecto a librerías.
- dpkg-genchanges lee la información de un árbol fuente Debian desempaquetado y construido, generando un fichero de control de los últimos cambios.
- dpkg-buildpackage es un script de control que se puede utilizar para automatizar la construcción del paquete.
- dpkg-distaddfile añade una entrada de un fichero a debian/files.
- dpkg-parsechangelog lee el fichero de cambios changelog) de un árbol fuente Debian

desempaquetado y genera una salida con la información de estos cambios, convenientemente preparada.

Advanced Packaging Tool (**APT**), es parte del sistema de gestión de paquetes creado por el proyecto Debian para simplificar la instalación y eliminación de programas en los sistemas GNU/Linux. Funciona como interfaz de línea de comandos para el conjunto de bibliotecas DPGK permitiendo acceder de forma sencilla a los más de 21.000 paquetes de software disponibles en los repositorios de Debian GNU/Linux. Su objetivo fundamental fue facilitar el proceso de gestión de programas en sistemas operativos que utilicen binarios *.deb* pero con el fin de adaptarse a la *LSB* se le añadieron funcionalidades que permiten utilizar programas distribuidos usando *RPM* (Red-Hat Package Manager)**.**

Actualmente funciona en otros sistemas operativos como Mac OS X (Fink) y OpenSolaris**.**

Aptitude es una interfaz para *APT* creada en 1999 y basada en una biblioteca *ncurses*, mediante la cual incorpora algunos elementos comunes a otras interfaces gráficas, como son los menús desplegables. Se creó para experimentar con un diseño más orientado a objetos, con el objetivo de que resultara un programa flexible y con características extensibles. Muestra una lista de paquetes de software y permite al usuario elegir de modo interactivo cuáles desea instalar o eliminar. Dispone de un poderoso sistema de búsqueda que facilita al usuario entender las complejas relaciones de dependencia que puedan existir entre los paquetes.

En el año 2000, se reescribió toda la interfaz gráfica creándose una nueva arquitectura, basada en la librería *libsigc++* y en conceptos de modernos *toolkits* de controles, como GTK+ y Qt. Éste cambio permitió que la interfaz ganara en facilidad de uso con nuevas características como los menús desplegables y cajas de diálogo emergentes.

Synaptic es un programa diseñado como interfaz gráfica del software *APT* para el sistema de gestión de paquetes de Debian GNU/Linux. Fue desarrollado utilizando las librerías GTK+ distribuidas por Gnome y usando el lenguaje Python.

Kynaptic es un programa que utiliza el código fuente base del Synaptic para realizar las mismas funciones que este en entornos de escritorio *KDE*.

Adept es otra interfaz gráfica de APT muy similar a Synaptic pero desarrollado utilizando las librerías *QT*, por lo que es altamente recomendado en entornos de escritorios *KDE*. Está compuesto por varios componentes que realizan funciones específicas:

- adept_updater: asistente de actualizaciones para el sistema.
- adept_batch: script para automatizar tareas mediante Adept.
- adept_manager: administrador de paquetes (similar a Synaptic).
- adept_installer: administrador de aplicaciones (Añadir/eliminar programas).
- adept_notifier: notificador de actualizaciones disponibles.

1.4.1.2. RPM, YUM e interfaces derivadas.

RPM es un sistema de gestión de paquetes originalmente desarrollado por *Red Hat* para la distribución Red Hat Linux. Los paquetes utilizados por el sistema tienen extensión *.rpm*, formato que actualmente es la base de la Base Estándar de Linux (LSB) y se emplean en una infinidad de distribuciones así como en otros sistemas operativos como Novell Net-Ware e IBM's AIX.

El sistema *RPM* tiene ventajas con respecto a otros sistemas de gestión de paquetes disponibles en el mercado, las más comunes son:

- Presenta al usuario una forma uniforme de instalar programas.
- El proceso de desinstalación es muy simple.
- Es muy popular, por lo que un repositorio de *RPM* contiene miles de aplicaciones de software.
- Como la instalación no es interactiva es muy fácil de automatizar.
- Se incluyen los códigos fuentes originales por lo que es muy fácil realizar verificaciones.
- Los paquetes pueden ser verificados criptográficamente con *GPG* y *MD5*.
- Los paquetes DeltaRPM, pueden actualizar un software previamente instalado sin necesidad del paquete original (función equivalente a un parche).

Como desventaja se puede decir que el sistema ha sido criticado por la falta de consistencia en el nombre de los paquetes y el contenido, por lo difícil que puede hacer el manejo automático de las dependencias.

Yellow dog Updater, Modified (**YUM**) es una utilidad de línea de comandos para efectuar la gestión de paquetes en sistemas GNU/Linux basados en el formato RPM. Fue desarrollado por Seth Vidal y un grupo de programadores voluntarios y se mantiene como parte del proyecto Linux@DUKE de la

Universidad de Duke. *YUM* es una utilidad totalmente reescrita a partir de Yellow Dog Updater y fue creada para actualizar y controlar los sistemas Red Hat utilizados en el Departamento de Física de la Universidad de Duke. Desde entonces ha sido adoptada por Fedora, CentOS, y otras distribuciones de GNU/Linux basadas en RPM, incluyendo el mismo Yellow Dog, donde reemplazó a la utilidad original YUP. El sistema de repositorios basados en YUM casi se ha convertido en un estándar internacional para la gestión de paquetes. Distribuciones como SUSE Linux han añadido soporte nativo a su sistema y lo utilizan en todos los repositorios de su versión libre OpenSUSE.

Yumex es una interfaz gráfica para el sistema YUM, desarrollada para su utilización en la distribución Fedora**.** Está escrito en el lenguaje Python por lo que su mantenimiento y reutilización son muy simples, lo que ha provocado que haya sido modificado en proyectos como Sabayon Linux para funcionar con otros sistemas de gestión de paquetes como Entropy.

Smart es una herramienta que persigue el ambicioso objetivo de proveer utilidades y algoritmos portables para solucionar adecuadamente los problemas de gestión de paquetes de software. Funciona en una gran cantidad de distribuciones de GNU/Linux, y provee según sus desarrolladores notables ventajas sobre herramientas nativas de las mismas (APT, APT-RPM, YUM, URPMI). Utiliza correctamente los formatos nativos de Debian (.deb), Red Hat (.rpm) y Slackware (tar.gz), aunque no recomienda el uso simultáneo de dichos formatos por la inestabilidad que puede provocar. Provee varías interfaces completamente integradas que permiten:

- Utilizar comandos en una terminal para actualizar, instalar, desinstalar, buscar, verificar y descargar paquetes desde Internet o un repositorio local.
- Interfaz de línea de comandos con completamiento de argumentos que hace más sencilla la realización de múltiples tareas.
- Interfaz gráfica muy sencilla y amistosa con respecto al usuario.
- Interfaz de línea de comandos con retroalimentación gráfica permitiendo integrar lo mejor de los dos modos de utilización.

URPMI es una herramienta de administración de paquetes que se encarga de instalar, eliminar, actualizar y hacer preguntas a los paquetes de software de aplicaciones locales o remotas. Envuelve al administrador de paquetes RPM extendiendo sus funcionalidades básicas. Trabaja con las fuentes oficiales de Linux Mandriva o con fuentes externas tales como las que otorga Penguin Liberation Front.

1.4.1.3. Entropy y Spritz

Entropy es la herramienta de gestión de paquetes desarrollada para Sabayon Linux por Fabio Erculiani, y probablemente uno de los proyectos más activos actualmente. Pretende agrupar las mejores prácticas y funcionalidades de otros gestores de paquetes como APT, RPM y Portage; de este último hereda el comportamiento básico. Está diseñado bajo una arquitectura cliente servidor que implementa una gran cantidad de funcionalidades extras a Portage y además mantiene la compatibilidad con el mismo por lo que es la mejor elección para basar el desarrollo de una distribución basada en Gentoo Linux. Se encuentra desarrollado en el lenguaje Python usando una filosofía estructurada aunque a sugerencia del Proyecto Nova está siendo modificado para ganar la potencia de una filosofía orientada a objetos y con el fin de hacer más sencilla su modificación, reutilización y extensión.

Está compuesto por los siguientes módulos independientes:

- Servidor
    - Entropy – Interfaz de línea de comandos que permite el acceso y maneja el resto de los módulos.
    - Reagent – Transforma la información de los metadatos de los paquetes binarios de Gentoo Linux en entradas de la Base de Datos de Entropy.
    - Activator – Realiza algunas pruebas de calidad a los archivos binarios y los añade al repositorio de una forma segura y consistente.
- Cliente
    - Equo – Interfaz de línea de comandos que incluye las librerías de interfaces a utiliza en caso de que se quiera desarrollar una aplicación gráfica. Permite efectuar el proceso de instalación, desinstalación y actualización de software en el sistema operativo así como otras muchas funcionalidades, como el cálculo de dependencias etc etc.
    - Interfaces GUI

Spritz es un programa muy joven (apenas se encuentra en su versión 0.03) que provee una interfaz visual a las funcionalidades de Equo. Está escrito en el lenguaje Python y reutiliza el código y aspecto de Yumex con algunas variaciones para adaptarse a las características particulares de Entropy.

Aunque todavía no es un desarrollo maduro debe mantenerse en observación por ser un software muy prometedor y por estar desarrollado por el equipo de Entropy lo que garantizará una excelente integración nativa con el sistema de gestión de paquetes que utiliza.

1.4.1.4. MSI

Instalador de Windows, es un motor para la instalación, mantenimiento y eliminación de programas en sistemas Microsoft Windows. La información de instalación y a menudo los archivos mismos, son agrupados en paquetes de instalación, bases de datos estructuradas como OLE Structured Storage Files (almacenamiento estructurado de ficheros), comúnmente conocido como "MSI files". Windows Installer es un avance significativo sobre su predecesor, Setup API, posee algunas características nuevas como la interfaz gráfica de usuario, la generación automática de la secuencia de desinstalación y un poderoso despliegue de capacidades, que lo hacen una alternativa viable a los instaladores independientes.

Estructura lógica de los paquetes:

Un paquete describe la instalación completa de un producto (Windows Installer no maneja dependencias entre productos) y esta universalmente identificado por una GUID. Un producto está compuesto de componentes y agrupados dentro de sus características.

Componentes:

Un componente es la mínima parte de un producto; cada componente es tratado por Windows Installer como una unidad. Los componentes pueden contener archivos, grupos de archivos, directorios, componentes COM, claves del registro de Windows, accesos directos y datos de otro tipo.

Los componentes están identificados globalmente por GUIDs, de forma que el mismo componente puede ser compartido entre varios del mismo paquete o de múltiples paquetes, idealmente a través del uso de la unión de módulos.

Rutas maestras (Key paths):

Una ruta maestra es un fichero específico, clave de registro, o fuente de datos ODBC que el autor del paquete especifica como crítico para un componente dado. Como las rutas maestras más utilizadas son en forma de fichero, se suele utilizar el término fichero maestro (key file). Un componente puede contener, a lo sumo, una ruta maestra; si un componente no tiene establecida de manera explícita

una ruta maestra, el directorio destino del componente es tomado como la ruta maestra. Cuando se ejecuta una aplicación basada en MSI, Windows Installer comprueba la existencia de estos ficheros críticos o claves de registro. Si existe un desajuste entre el estado actual del sistema y el valor especificado en el paquete MSI, entonces la característica asociada es reinstalada.

1.4.2. Sistemas de instalación basados en código fuente.

1.4.2.1. Porticus y Macport

MacPorts es una herramienta de código abierto creada para gestionar paquetes de software en los sistemas operativos de la familia Mac OS X. Está diseñada para compilar, instalar y actualizar software desde una interfaz de línea de comandos, Aqua o X11 por lo que facilita y simplifica el acceso a aplicaciones de software libre creadas por proyectos como GNU a los usuarios de sistemas operativos Mac. Se encuentra distribuido bajo licencia BSD y accede a repositorios de aplicaciones libres cuyo número asciende a 4664 en el momento actual.

Porticus es una interfaz gráfica para MacPorts, provee soporte para actualizaciones tipo Sparkle y una biblioteca de scritps de Apple. Se caracteriza por ser ligero, sencillo y distribuido bajo licencias libres. Permite búsquedas descriptivas, detección de actualizaciones así como acceso visual y simple a la mayoría de las funcionalidades de MacPorts.

1.4.2.2. Portage e interfaces derivadas.

Portage es un sistema de distribución de software inspirado en Ports de BSD. Consiste en un árbol local, que contiene las descripciones de los paquetes de software así como los ficheros receta con la información necesaria para instalarlos. Está escrito en el lenguaje Python y es una de las características distintivas de la distribución Gentoo Linux de la cual es la piedra angular. Fue diseñado para trabajar con distribuciones de códigos fuentes de forma nativa por lo que posee la capacidad de descargar, compilar y generar los archivos ejecutables y la documentación correspondiente. Portage permite la optimización a nivel de hardware y software a la hora de compilar los paquetes, permitiendo crear sistemas a la medida o de muy buen rendimiento. Además permite la utilización de paquetes precompilados tanto nativos como de otras distribuciones como RPM y .deb. Está considerado junto con el sistema Conaris como una evolución muy significativa dentro de los sistemas de gestión de paquetes en sistemas GNU/Linux.

A pesar de tener muchas características interesantes y útiles, Portage tiene varios problemas que hacen su utilización muy molesta para usuarios con poca experiencia:

- La gestión de paquetes desde los repositorios alternativos de binarios es muy lenta.
- El proceso de compilación tarda horas en completarse.
- Al examinar su código no es difícil notar que está constituido por una colección de scripts que realizan su cometido sin una arquitectura de software coherente y sólida lo que trae serios problemas de rendimiento.
- Está orientado a usuarios con conocimientos avanzados de GNU/Linux.

Porthole es una interfaz gráfica para Portage desarrollada utilizando librerías GTK y el lenguaje de programación Python. Está basada en un proyecto descontinuado llamado Gportage y fue diseñada para ejecutarse de forma nativa en entornos Gnome, aunque funciona correctamente también en entornos KDE. Provee una vista jerárquica del árbol de paquetes que permite hacer búsquedas y acceder a todas las funcionalidades de Portage heredando su bajo rendimiento.

Portato es una interfaz visual de Portage creada para facilitar el trabajo de los usuarios en determinadas situaciones, no pretende sustituir a la interfaz de línea de comandos sino trabajar en combinación con esta. Permite acceder a la mayoría de las funcionalidades de Portage, sus principales características son:

- Escrito en Python por lo que puede hacer llamadas directas a la API de Portage sin tener que parsear la salida de los comandos.
- No tiene base de datos propia, utiliza la de Portage directamente.
- Trabaja de forma muy inteligente los ficheros de configuración.
- Tiene un sistema de plugins que lo hacen extensible.

1.4.3. Sistemas de metapaquetes.

Los sistemas de **metapaquetes** son aquellos que unifican la gestión de paquetes en varías distribuciones de GNU/Linux bajo un mismo formato sin tener en cuenta el formato nativo del sistema. Algunos ejemplos son:

Klik proporciona una forma sencilla de instalar paquetes de software para la mayor parte de

distribuciones sin los problemas de dependencias tan comunes en otros formatos de paquetes. Ofrece a los usuarios un listado de aplicaciones que pueden ser instaladas mediante el navegador Firefox. Estas aplicaciones provienen principalmente de Debian GNU/Linux, pero funcionan en una gran variedad de distribuciones actuales. Cuando hacemos click para instalar una aplicación el sistema descargará lo necesario y construirá un archivo llamado "*.cmg" el cual podrá ser ejecutado por el usuario y que tendrá contenido internamente todas las dependencias requeridas para ejecutar el programa. Klik maneja un concepto muy revolucionario a la hora de gestionar las aplicaciones y con un excelente futuro pero actualmente varias desventajas importantes que limitan que se generalice su utilización, las más comunes son:

- Requiere permisos de root para instalar el sistema, pues debe ser modificado el fichero /etc/fstab, lo cual puede traer riesgos de seguridad.
- En el listado web no se muestra la versión de la aplicación que se va a instalar.
- Mala integración con entornos Gnome, para ejecutar los ficheros .cmg hay que ejecutar desde nuestro directorio personal: "./.zAppRun Desktop/aplicacion.cmg".
- No funcionan todas las aplicaciones disponibles.

Autopackage es un sistema de gestión de paquetes relativamente nuevo para GNU/Linux, que provee un empaquetado casi universal capaz de funcionar en todas las distribuciones de GNU/Linux. Ofrece a los desarrolladores una forma de empaquetar su aplicación, de forma que esta sea instalable en prácticamente cualquier distribución actual, el usuario solo tendría que descargar un fichero llamado "*.package", que al ser ejecutado por primera vez instalará el sistema y las sucesivas aparecerá directamente el asistente para la instalación. Los programas son instalados sin privilegios de administración por lo que no se compromete la seguridad del sistema y un usuario promedio e inexperto puede agregar o eliminar programas sin ninguna dificultad. Autopackages no aspira a sustituir los sistemas nativos de gestión de paquetes sino que pretende actuar como un complemento para aplicaciones no relacionadas directamente con el núcleo del sistema. Se integra correctamente con entornos KDE, Gnome e incluso a nivel de terminales de sistema.

1.5. Metodologías y tecnologías comúnmente empleadas.

En éste epígrafe se realiza un análisis de las tecnologías y metodologías más comunes empleadas en el desarrollo de las herramientas de gestión de paquetes descritas anteriormente. Se pretende,

bajo un análisis comparativo, decidir cuales son las más adecuadas para ser utilizadas en el gestor de paquetes de Nova.

1.5.1. C ++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980 como extensión del lenguaje de programación C. Abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos. Sus principales fortalezas radican en las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica.

Posee además una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores).
- Identificación de tipos en tiempo de ejecución (RTTI).

C++ está considerado por muchos como el lenguaje más potente de todos, debido a que permite trabajar tanto a alto como a bajo nivel pero es uno de los lenguajes más difíciles de aprender y mantener el código lo que limita mucho su empleo.

1.5.2. Python

Python es un lenguaje de programación limpio y elegante que permite dividir el programa en módulos reutilizables. Incluye una gran colección de módulos estándar que se pueden utilizar como base de los programas que proporcionan funcionalidades para el manejo de ficheros, llamadas al sistema, sockets y hasta interfaces a librerías gráficas como (GTK, QT, TK). Es un lenguaje interpretado, característica que ahorra un tiempo considerable en el desarrollo del programa, al no ser necesario compilar ni enlazar. Es un lenguaje multiparadigma por lo que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación estructurada y programación funcional. Soporta otros muchos paradigmas mediante la utilización de extensiones las cuales pueden ser escritas fácilmente en otros lenguajes como C y C++ o desde el mismo Python. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos). Es uno de los lenguajes mas sencillos de aprender,

el código generado es simple y legible por lo que mantener y desarrollar una aplicación en Python es una tarea realmente sencilla.

1.5.3. GTK

GTK+ (GIMP ToolKit) es un grupo de bibliotecas o rutinas para desarrollar interfaces gráficas de usuario (GUI) para principalmente los entornos gráficos GNOME, XFCE y ROX de sistemas GNU/Linux. Está distribuida bajo licencia LGPL y tiene características multiplataforma por lo que es parte importante del proyecto GNU. Inicialmente fue creada para desarrollar el programa de manejo de imágenes GIMP, pero actualmente es muy se ha generalizado y extendido su utilización en muchos programas. Permite programar con C, C++, C#, Java, Perl, PHP o Python aunque el número de lenguajes soportados se incrementa día a día.

Se basa en varias bibliotecas del equipo de GTK+ y GNOME:

- **GLib**. Biblioteca de bajo nivel y estructura básica de GTK+ y GNOME. Proporciona el manejo de estructuras de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución como ciclos, hilos, carga dinámica o un sistema de objetos.
- **GTK**. Biblioteca que contiene los objetos y funciones para crear la interfaz de usuario. maneja todos los controles *(widgets)* visuales, ventanas, botones etc.
- **GDK**. Biblioteca que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.
- **ATK**. Biblioteca para crear interfaces de gran accesibilidad importante para personas discapacitadas o minusválidas. Pueden usarse útiles como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o mouse.
- **Pango**. Biblioteca para el diseño y renderizado de texto, se enfoca especialmente en la internacionalización del código. Es el núcleo para manejar las fuentes y el texto de GTK+2.
- **Cairo**. Biblioteca de renderizado avanzado de controles de aplicación.

1.5.4. QT

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario creada por la compañía noruega Trolltech y utilizada en KDE, un entorno de escritorio para sistemas GNU/Linux y BSD entre otros. Utiliza el lenguaje de programación C++ de forma nativa aunque existen enlaces soportados para C, Python (PyQt), Java (Qt Jambi), Perl (PerlQt), Gambas (gb.qt), Ruby (QtRuby), PHP (PHP-Qt) y Mono (Qyoto). La *API* de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML y para el manejo de ficheros y además de estructuras de datos tradicionales.

Se encuentra disponible para las siguientes plataformas:

- **X11** - Para X Window System con licencia GPL. (Linux, Unix, BSD).
- **Mac** - Para Mac OS X bajo la licencia GPL.
- **Windows** - Para sistemas Windows con licencia GPL (las versiones anteriores a la 4.X eran privativas para este sistema operativo).
- **PDA** - Para dispositivos empotrados, también bajo licencia GPL y normalmente distribuido junto a Qtopia, un entorno completo para *PDAs.*

Existen 4 ediciones de Qt disponibles dentro de cada una de las plataformas anteriores, llamadas:

- **Qt Console** - Edición para desarrolladores non-GUI.
- **Qt Desktop Light** - Edición con nivel reducido de GUI, orientado a redes y bases de datos.
- **Qt Desktop** - edición completa.
- **Qt Open Source Edition** - Edición "completa", con algunas excepciones como el control ActiveQt (ActiveX) para Windows, destinada a desarrolladores de software libre.

1.5.5. Programación Extrema

La programación extrema es un enfoque de la ingeniería de software formulado por Kent Beck y constituye la más destacada de las metodologías ágiles de desarrollo de software. Se diferencia de las metodologías tradicionales principalmente en que hace más énfasis en la adaptabilidad del que en

la previsibilidad. Considera que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos y promueve que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo e invertir esfuerzos después en controlar los cambios en los mismos. Se puede considerar la programación extrema como la adopción de las mejores prácticas de desarrollo de software y su aplicación de de manera dinámica durante el ciclo de vida del software.

Las características fundamentales de la metodología son:

- Desarrollo iterativo e incremental.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación en parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto.
- Frecuente integración del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto.
- Simplicidad en el código: Es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo en caso necesario, que realizar algo complicado y quizás nunca utilizarlo.

1.6. Conclusiones del capítulo

Después de realizar un estudio de los programa de gestión de paquetes descritos anteriormente se ha llegado a la conclusión de que tienen las mismas funciones en su generalidad, solamente se

diferencian en detalles de como están implementados o de las tecnologías utilizadas, pero el núcleo de funcionalidades es muy similar en todas las herramientas.

Nova es una distribución basada en Gentoo Linux, que hereda su sistema de paquetería y los binarios generados por Portage, y que se desea mantener la compatibilidad con la distribución madre; El equipo de desarrollo está compuesto mayormente por estudiantes lo que hace necesario utilizar tecnologías fáciles de asimilar en tiempos cortos y metodologías ágiles de desarrollo que permitan liberar las versiones de las herramientas con la rapidez característica de los proyectos de software libre con los cuales se compite a nivel internacional.

Partiendo de la afirmación anterior y teniendo en cuenta las características particulares del Proyecto Nova, se decide reutilizar el sistema de gestión de paquetes Entropy de Sabayon Linux, para crear el de Nova empleando el lenguaje de programación Python con bibliotecas GTK y la metodología de desarrollo Programación Extrema con el fin de desarrollar una herramienta de gestión de paquetes que satisfaga las necesidades de la distribución y de los usuarios del sistema.

2. Sistema gestor de paquetes de Nova

2.1. Introducción

En este capítulo se presenta la situación actual del proyecto Nova, sus necesidades y problemas críticos que provocan la situación problémica de la presente investigación. Se explica como funciona el proceso de desarrollo de los productos liberados por el proyecto y las características y expectativas de los usuarios potenciales a los cuales se desea impactar. Se describe además la solución propuesta y se planifica su desarrollo, haciendo hincapié en la planificación de las iteraciones, plan de entrega y las fases iniciales del proceso de ingeniería de software.

2.2. La plataforma Nova.

La idea de crear una distribución de GNU/Linux surge de un grupo de estudiantes de la Universidad de las Ciencias Informáticas, como respuesta a la necesidad de una plataforma que garantizara la compatibilidad del software que se estaba desarrollando, con los sistemas libres que tanto auge tienen en el mundo. Posteriormente el proyecto se volvió más ambicioso con la inminente migración a software libre de algunos organismos estatales y ministerios, convirtiéndose en una plataforma para generar distribuciones a la medida.[3]

El equipo de trabajo inicialmente se dedicó a hacer una personalización estable de *Gentoo*, una distribución poco recomendada a usuarios sin experiencia por su alta complejidad, pero perfecta para la creación de un nuevo sistema personalizado por su alta flexibilidad ante diferentes entornos y por su orientación al trabajo con código fuente en lugar de paquetes precompilados.[4] Gentoo brinda las herramientas de software necesarias para establecer políticas de desarrollo que permiten mantener un equilibrio adecuado entre las facilidades de utilización y el rendimiento, algo muy controversial en los sistemas operativos. Con el tiempo el proyecto fue modificando sus objetivos y razón de ser, evolucionando de un sistema operativo personalizado con núcleo Linux a una plataforma para generar sistemas operativos de forma dinámica que cumplan las especificaciones requeridas por cada uno de los clientes.

En la actualidad el Proyecto Nova se encuentra en una etapa de despliegue de los productos generados, los cuales deben llegar a un grupo de usuarios que se han identificado como potenciales

por encontrarse en una etapa de migración de tecnologías y a la comunidad cubana de software libre. Para vencer la resistencia al cambio de los primeros y el escepticismo de los segundos, es necesario proveer una distribución de GNU/Linux con calidad y un conjunto de características indispensables:

- Instalación y actualización sencilla del sistema.
- Alta disponibilidad de programas de software.
- Instalación, actualización y desinstalación de software de manera simple e intuitiva.
- Actualizaciones de seguridad liberadas regularmente y fácilmente aplicables.

Como se observa, los requisitos antes expuestos son una pequeña muestra de lo que debe hacer el sistema operativo y se relacionan directamente con el sistema de gestión de paquetes de Nova, el cual debe garantizar cada una de esas funcionalidades, y hacerlo de manera que un usuario inexperto sea capaz de beneficiarse de ellas sin mucho esfuerzo.

2.3. Procesos internos de construcción de un sistema Nova.

El proceso de construcción y desarrollo de un sistema Nova funciona como una línea de ensamblaje continua soportada por una infraestructura tecnológica de vital importancia de donde se obtienen los componentes necesarios para generar un sistema Nova. Dicha infraestructura tecnológica está compuesta por:

- Un repositorio (gentoo.prod.uci.cu) de donde se descargan los códigos fuentes necesarios para construir los programas que tendrá la distribución.
- Un *overlay* (http://svn.nova.prod.uci.cu/svn/nova-overlay) de donde se obtienen los códigos fuentes de los programas que Nova mantiene de forma autónoma.
- Un repositorio de paquetes precompilados listos para instalar a través de Portage, el sistema de gestión de paquetes de Gentoo Linux.
- Un repositorio de paquetes precompilados que funciona utilizando Entropy y fue creado recientemente para asimilar el nuevo gestor de paquetes en la distribución.

Para que el proceso de construcción de Nova concluya exitosamente debe pasar por seis etapas o fases que conforman la línea de producción y cumplen funciones muy específicas en el desarrollo del producto final las cuales se describen a continuación haciendo hincapié en las relacionadas

directamente con la investigación:

1. Etapa de construcción de binarios

   Es la fase en la cual se crean los archivos instalables de las aplicaciones disponibles para incluir en el sistema. Se eligen los programas que se insertarán en la nueva versión de Nova para compilarlos desde el repositorio de códigos fuentes. Se caracteriza por un proceso intensivo de compilación e instalación, procesando cerca de 4000 paquetes de software que serán agregados en forma de ficheros binarios al repositorio de paquetes precompilados donde se almacenan hasta la etapa de ensamblaje.

2. Etapa de construcción del *sistema base*

   Fase donde se crea una versión mínima del sistema operativo, sin interfaz gráfica y con la cantidad de programas indispensables para funcionar. Dicha versión mínima será los cimientos de donde evolucione cualquier versión de Nova que se requiera desarrollar. Durante el transcurso de esta etapa se realiza un proceso de compilación e instalación de programas no tan extenso como durante la construcción de binarios, pero se procesan paquetes de software vitales para el sistema, los cuales no pueden tener ningún defecto pues se afectaría gravemente la estabilidad del sistema.

3. Etapa de compilación del núcleo

   Se construye el núcleo con los soportes y características definidas por el cliente, esta etapa es el primer paso de personalización en caso de una solución a la medida o una versión de servidores.

4. Etapa de ensamblaje

   La etapa de ensamblaje es la fase que concierne más directamente al problema que da motivo a la investigación. Esta etapa se caracteriza por un proceso intensivo de instalación de software en el sistema que se va a liberar. Se toma el sistema base, se le pone manualmente el núcleo y se procede a instalar los paquetes de software que están almacenados en el repositorio de binarios. El sistema fruto del proceso queda marcado como versión alfa, y posee gran parte de las funcionalidades contenidas en los requerimientos del cliente.

5. Etapa de configuración

   Configuración es el proceso en el cual se deja el sistema en el estado en el cual será utilizado,

se establecen los permisos y se crean las políticas de acceso, además de todo el diseño de interfaz gráfica e identidad del producto. Dichas tareas son muy engorrosas y son realizadas manualmente por los desarrolladores, lo cual conlleva una perdida de tiempo y eficiencia y tiempo en el proceso.

6. Etapa de pruebas

   La etapa de pruebas, es la fase en la cual se prueba el funcionamiento del producto final, se reportan y corrigen "*bugs*" del sistema y se establecen varías iteraciones hasta que el sistema operativo esté completamente libre de errores.

Como se puede observar, en cada una de las etapas del proceso se construyen piezas independientes que posteriormente conformarán el sistema; piezas que son creadas y ensambladas mediante un gestor de paquetes (Portage en éste caso). En las diferentes fases hay tareas que se realizan manualmente afectando el rendimiento y la calidad del proceso de construcción pues un error humano puede provocar defectos en el producto, es necesario optimizar dicho proceso y optimizar los puntos débiles para lograr productos finales con más calidad.

2.4. Problemas del proceso de gestión de paquetes de Nova

Los problemas de paquetes de software en los sistemas Nova se dividen en dos aristas independientes: los problemas que enfrentan los usuarios finales durante la utilización del sistema y los que enfrentan los desarrolladores durante el proceso de construcción.

Los usuarios finales de un sistema operativo esperan que las nuevas funcionalidades, programas, actualizaciones y demás servicios del proyecto se encuentren fácilmente asequibles, al alcance de unos pocos click del ratón. La idea de un gestor de paquetes como Portage, que como se ha descrito está orientado al trabajo con los códigos fuentes, desarrollado por una equipo de "hackers" y para "hackers" , con una interfaz compleja y un rendimiento que deja mucho que desear no es una opción popular entre la comunidad de potenciales usuarios de Nova. Un usuario promedio no tiene un conocimiento avanzado de sistemas operativos GNU/Linux ni el tiempo necesario para construir su propio software evitando las incompatibilidades que pueden surgir durante ese proceso; en la mayoría de los casos detesta trabajar con una interfaz de línea de comandos, prefiriendo una interfaz gráfica que provea todo lo que necesite, que sea capaz de notificar cuando hayan actualizaciones disponibles y de obtenerlas en el mismo instante en que son liberadas sin esfuerzo patente.

Evidentemente, Portage y por tanto Nova, no satisfacen las expectativas de un usuario inexperto, público potencial al cual está dirigido el producto por lo que en caso de una salida al mercado puede provocar un estado de opinión negativo que afecte el grado de aceptación del sistema en caso de un despliegue a gran escala.

Por la arista del equipo de desarrollo los problemas son muy diferentes, el proceso de construcción de los sistemas Nova debe realizarse varias veces anualmente, los recursos humanos de los cuales dispone el Proyecto para hacerlo son escasos por lo que es necesario lograr una manera de que los desarrolladores contribuyan a las innovaciones y a crear conocimiento e inteligencia dentro de la organización en vez de a realizar tareas monótonas y repetitivas. En otras palabras, es necesario automatizar todos los procesos y tareas que se realizan manualmente en estos momentos, para enfocar el esfuerzo del desarrollador en actividades mucho más productivas.

Durante las etapas de ensamblaje y configuración un Especialista en Sistemas tiene la responsabilidad de incluir en el sistema el núcleo y toda la identidad visual del producto así como la configuración de la interfaz gráfica de usuario, para esto necesita copiar manualmente los diferentes componentes necesarios, modificar cada uno de los archivos de configuración de los programas implicados así como realizar trabajos de limpieza de logs, historiales etc, para evitar que durante el proceso de liberación queden rastros de sus acciones. Las responsabilidades antes descritas son fácilmente automatizables y la forma mas lógica o idónea de hacerlo es a través del sistema de gestión de paquetes, pues solo sería necesario publicar los componentes en el repositorio de Nova con la información necesaria para que la interfaz de línea de comandos configure automáticamente el gestor de arranque y el entorno de escritorio, durante las acciones posteriores a la descarga e instalación de los correspondientes paquetes. De esta manera todas las tareas de dichas fases se vuelven transparentes al desarrollador, el cual solo con instalar los paquetes correspondientes hará su trabajo con calidad e incluso puede ser sustituido por un miembro del equipo menos experimentado en caso de necesidad.

El segundo problema está dado por el comportamiento de la herramienta Portage a la hora de instalar paquetes precompilados. Al no estar diseñado para esta función, su comportamiento es errático algunas veces, sobre todo a la hora de detectar las dependencias de un programa a instalar. Portage tiende a ignorar la existencia de algunos paquetes no especificados explícitamente en el *ebuild* del programa requerido, comprometiendo el funcionamiento del mismo y la estabilidad del sistema. Además el tiempo requerido para descargar e instalar un programa desde un repositorio en Internet

es muy alto, Portage realiza un chequeo de metadatos en el servidor, que demora en dependencia de la cantidad de paquetes disponibles, lo cual provoca una perdida grande de tiempo para el responsable de incluir el software requerido en el sistema durante la Etapa de ensamblaje. Por lo antes expuesto se pretende sustituir Portage en todo el proceso de construcción de Nova, exceptuando la Etapa de construcción de binarios por un sistema de gestión de paquetes más eficiente y rápido que cumpla con los requerimientos del equipo de desarrollo.

2.5. Descripción de la solución propuesta

El desarrollo del sistema informático propuesto como solución lleva dos pasos fundamentales: la asimilación e instalación de un sistema Entropy y el desarrollo de un programa bautizado Summon que actúe como interfaz gráfica para facilitar la interacción con el usuario.

Entropy provee todas las funcionalidades de la parte del servidor (repositorio) y la interfaz de línea de comandos en los sistemas cliente (equo) dando solución a los problemas críticos detectados en el proceso de construcción. Mejora el rendimiento y la eficiencia del equipo de desarrollo al eliminar las demoras provocadas por la herramienta Portage; mediante la creación de metapaquetes puede manejar la gestión de binarios del núcleo y de todos los elementos de diseño de forma independiente, facilitando el proceso de mantenimiento del árbol de software disponible de Nova.

La herramienta a diseñar en la presente investigación debe permitir a los usuarios de los sistemas Nova gestionar los programas que desean utilizar en sus computadoras mediante una interfaz gráfica sencilla y consistente con el entorno de escritorio Gnome predeterminado en la distribución. Permitirá además gestionar los diferentes repositorios Entropy de donde obtener el software y sincronizar con los mismos fácilmente, realizar búsquedas avanzadas entre los paquetes disponibles e informar a los usuarios de los errores ocurridos generando un reporte del error detallado.

Para proveer las funcionalidades antes descritas sin comenzar un gestor de paquetes desde cero, Summon pretende reutilizar o re-implementar funcionalidades y bibliotecas del sistema cliente de Entropy sin afectar o reescribir el código original. Para lograr éste propósito se dividirá la aplicación en 3 módulos:

1. **Módulo de Gestión de Paquetes:** Módulo encargado de realizar las tareas de instalación, actualización o desinstalación de paquetes de software.
2. **Módulo de Configuración:** Módulo responsable de manejar las configuraciones internas de

la herramienta Summon, guardar las opciones seleccionadas y permitir algunas tareas de optimización de rendimiento.

3. **Módulo de Gestión de Repositorios:** Módulo encargado de implementar las funcionalidades de añadir, modificar y eliminar repositorios, así como la sincronización del sistema con los mismos.

2.6. Instalación y configuración de un servidor Entropy

Para asimilar el sistema Entropy se debe partir sistema Nova en el cual se instalará el paquete *sys-apps/entropy-server utilizando* el gestor de paquetes deseado. Posteriormente se debe acceder a */etc/entropy* donde se encuentran los ficheros de configuración necesarios:

- **entropy.conf**

  Configuraciones generales, tales como el nombre de la distribución, proxy http y ftp a utilizar para acceder al repositorio .

- **etc-portage-bashrc**

  Fichero que garantiza que al utilizar el sistema Portage de Gentoo Linux para instalar o desinstalar paquetes se actualice la base de datos de entropy

- **packages/license.mask**

  Archivo utilizado para bloquear paquetes cuyas licencias no se desea tener en el repositorio.

- **packages/package.keywords**

  Para estabilizar paquetes en fase de pruebas (testing).

- **packages/package.mask**

  Para evitar la instalación de algunos paquetes.

- **packages/package.unmask**

  Para estabilizar paquetes inestables (masked).

- **packages/packages.db.lic_whitelist**

  Para no adicionar a la base de datos paquetes con determinada licencia.

- **packages/packages.db.mask**

  Para evitar la adición de algunos paquetes al repositorio.

- **packages/packages.db.repo_updates**

  En este archivo se configura como se realizarán los movimientos de paquetes en los repositorios, de una categoría a otra, cambios en los slots, etc.

- **repositories.conf**

  Archivo de configuración de repositorios, donde se definen los diferentes repositorios que estarán disponibles.

- **socket.conf**

  Permita configurar un sistema para administrar sus paquetes por remoto, mediante políticas del servidor Entropy.

- **activator.conf**

  Archivo de configuración de la herramienta activator, donde se puede definir el tipo de compresión de la Base de Datos.

- **reagent.conf**

  Archivo principal de configuración de la herramienta reagent donde se definen los RSS y sus correspondientes direcciones de Internet.

- **server.conf**

  Archivo donde configuramos nuestro acceso para escribir en el repositorio, por ejemplo usando el protocolo ftp.

Como se observa Entropy provee muchas funcionalidades a un administrador del repositorio por lo que su configuración no será descrita detalladamente, pues depende de las políticas de la institución donde se vaya a emplear. La configuración del repositorio de Nova a implementar posee las características siguientes: Permite acceso FTP y HTTP a dos repositorios diferentes (estable y estándar), habilitando la compatibilidad con el Portage de Gentoo Linux y utilizando un RSS para notificar cuando hay algún cambio en los paquetes disponibles. No se hará ningún tipo de filtrado por licencia ni se bloquearan paquetes, solo se añadirán programas cuyo funcionamiento esté garantizado.

Teniendo el repositorio en funcionamiento, se tiene la base para poder implementar la gestión de paquetes de cara al cliente, construyendo la interfaz que utilizarán los usuarios para instalar, actualizar o desinstalar paquetes en sus computadoras personales o estaciones de trabajo.

2.7. Desarrollo de la herramienta Summon.

El desarrollo de Summon, el gestor de paquetes gráfico de NOVA es la segunda parte de la solución propuesta. Constituye el complemento de cara al usuario del servidor Entropy asimilado anteriormente, y es el software responsable de ofrecer al cliente las funcionalidades disponibles de forma sencilla.

2.7.1. Fase de planificación de Summon

La metodología programación extrema tiene 4 fases fundamentales: Planificación, Diseño, Desarrollo, y Prueba; de las cuales serán descritas las 2 primeras en el presente documento.

XP plantea la planificación como un diálogo entre el desarrollador y el cliente los cuales llegarán a un consenso con respecto al alcance del proyecto, la prioridad a la hora de implementar las funcionalidades requeridas, la composición de las versiones y la fecha de entrega de las mismas. La metodología plantea una serie de liberaciones parciales al terminar cada iteración para retroalimentar el proceso de desarrollo y probar si se están cumpliendo los requisitos del cliente.

Los principales artefactos de esta fase son las historias de usuario, que sirven de guía a todo el proceso de desarrollo. Son utilizadas para especificar los requisitos del software y las escriben los propios clientes según su percepción de las necesidades del sistema. Las historias de usuario proporcionan los detalles sobre la estimación del riesgo y tiempo que llevará la implementación de determinadas funcionalidades. Son una vista a muy alto nivel del costo y esfuerzo del proyecto que se desea desarrollar.

2.7.1.1. Historias de usuario

Para la implementación del gestor de paquetes se realiza la redacción de las historias de usuario según las necesidades identificadas por un equipo de desarrollo del proyecto realizará la función del cliente durante todo el tiempo que dure la implementación de la herramienta y su posterior evolución.

Dicho equipo es totalmente ajeno al colectivo que se encargará de desarrollar la herramienta.

Las historias de usuario a implementar son:

- Gestionar paquetes.
- Gestionar repositorios.
- Sincronizar con repositorio.
- Mostrar paquetes.
- Buscar paquete.

Historia de Usuario	
Fecha:	**Versión**:1.0
Número: 1	**Dependencia:** HU-2
Nombre historia: Gestionar paquetes.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Descripción: El sistema procesará la petición de acciones referentes a paquetes de software. Se procederá a instalar, desinstalar o actualizar los paquetes que se encuentren en la cola de tareas seleccionadas por el usuario.	
Observaciones: En caso de los procesos de instalación y actualización, el sistema operativo debe tener acceso via Internet o red local al repositorio de paquetes, o estos deben haber sido copiados manualmente a la computadora donde serán instalados.	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 2	**Dependencia:** HU-3
Nombre historia: Sincronizar con repositorios	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: El sistema conecta con el servidor y procede a descargar la última base de datos de paquetes disponibles en el repositorio.	
Observaciones: El sistema operativo debe tener acceso a una red con conexión al servidor donde se encuentra el repositorio.	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 3	**Dependencia:** No
Nombre historia: Gestionar repositorios	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: Se insertan los datos de los diferentes servidores y repositorios de donde se obtendrán los paquetes, dichos datos serán almacenados en los correspondientes archivos de configuración del sistema.	
Observaciones:	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 4	**Dependencia:** No
Nombre historia: Mostrar paquetes instalados	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: Se realiza una consulta a la base de datos de los paquetes instalados para extraer una lista del software disponible en el sistema, se muestra en la vista de paquetes a eliminar.	
Observaciones: La base de datos debe haber sido generada anteriormente utilizando Equo.	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 5	**Dependencia:** HU- 3
Nombre historia: Mostrar paquetes disponibles	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: Se realiza una consulta a la base de datos de los paquetes disponibles, para extraer un listado del software disponible en el repositorio. Se muestra dicha lista en la vista de paquetes a instalar.	
Observaciones: Debe haberse sincronizado previamente con el repositorio.	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 6	**Dependencia:** HU-3, HU- 4
Nombre historia: Mostrar paquetes actualizables	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Descripción: Se obtiene una lista de paquetes cuya versión instalada en el sistema es menor que la disponible en el repositorio. Se muestra dicha lista en la vista de actualizaciones.	
Observaciones: Debe haberse sincronizado previamente con el repositorio y generado la base de datos de paquetes instalados.	

Historia de Usuario	
Fecha:	**Versión:**1.0
Número: 7	**Dependencia:** HU-3, HU-4, HU-6
Nombre historia: Mostrar información de un paquete	
Prioridad en negocio: Medio	**Riesgo en desarrollo:** Bajo
Descripción: Mostrará información de un paquete determinado (licencia, descripción), extrayéndola de la base de datos correspondiente a la vista en la que se encuentre el programa.	
Observaciones: Debe haberse sincronizado previamente con el repositorio y generado la base de datos de paquetes instalados.	

2.7.1.2. Planificación de las Historias de Usuario.

Planificar acertadamente el proyecto es necesario estimar de forma ideal el tiempo que les tomará a los programadores la codificación de cada una de las historias de usuario. Las condiciones ideales se dan cuando se escribe el código sin distracciones y con una dedicación de tiempo completo. La estimación debe dar un máximo de 2 semanas, si es necesario mas tiempo debe considerarse una división de la historia de usuario en partes mas pequeñas. Como es lógico, debido a las características del equipo de desarrollo y de la UCI en general, el tiempo de estimación ideal de las historia del gestor de paquetes de Nova puede extenderse un poco más por cargas docentes y un margen de fallo debido a actividades imprevistas.

A partir de la prioridad de las historias se decide cuales de ellas se implementarán en las primeras iteraciones, las funcionalidades críticas del sistema deben ser codificadas en iteraciones tempranas del ciclo.

No	Nombre HU	Prioridad	Riesgo	Esfuerzo (Días)	Iteración
1	Gestionar paquetes	Alta	Alto	15	1
2	Gestionar repositorios	Media	Bajo	15	3
3	Sincronizar con repositorio	Alta	Medio	5	1
4	Mostrar paquetes	Alta	Medio	25	2
5	Buscar paquete	Baja	Bajo	5	3

2.7.1.3. Plan de Entregas

- **Iteración 1**: Se propone codificar las historias de usuario más importantes, que proveen las funcionalidades mas críticas del sistema: #1 “Gestionar Paquetes” y #3 “Sincronizar con repositorios”.
- **Iteración 2**: Se codificará una historia de prioridad alta y riesgo medio, con mucha interacción con el usuario: #4 “Mostrar paquetes.
- **Iteración 3**: Última iteración, se desarrollarán las historias de menos complejas y que representan menos el interés del cliente: #2 “Gestionar repositorios” y #5 “Buscar paquete”.

2.7.2. Modelación de las historias de usuario

Las historias de usuarios son el principal artefacto de la metodología XP pues describen los requerimientos que debe cumplir. Por éste motivo y para ganar claridad en el proceso de desarrollo y comprensión del dominio del problema a resolver es muy importante modelarlas apoyándose en diagramas que ilustren el funcionamiento del sistema.

La herramienta objetivo de la presente investigación será descrita utilizando técnicas de modelado ágil y lenguaje UML para asegurar la total comprensión de las historias de usuario.

2.7.2.1. Modelo del Dominio

El modelo del dominio se puede definir como un modelo conceptual que describe las principales entidades presentes y sus relaciones; usualmente es capaz de identificar los métodos y atributos de las entidades fundamentales, por lo que se puede decir que provee una primera vista estructural del sistema. Es creado para documentar los conceptos claves y el vocabulario que será utilizado por el equipo de desarrollo al referirse al sistema, y puede ser utilizado para garantizar o aumentar la comprensión del problema por los propios clientes (stakeholders), por lo que es muy útil como herramienta de comunicación en caso de que el proyecto implemente una división en equipos técnicos y de negocio. (Ver Anexo 1. Modelo del Dominio)

2.7.2.2. Iteración 1

En la primera iteración se deben implementar las características principales del sistema, para poder liberar una primera versión o prototipo funcional al cliente. Como la lógica del negocio de la herramienta que se quiere implementar está en las historias #1 y #3, estas fueron las que se seleccionaron para realizarlas en la primera iteración.

La historia #1 se dividió en tres tareas de la ingeniería fundamentales, que proveerán las funcionalidades básicas de instalar, desinstalar y actualizar paquetes de software, el núcleo de la herramienta Summon. (Ver Figura 2.1)

Tarea de Ingeniería	
Número tarea:1	**Número historia:**1
Nombre tarea: Instalar Paquete	
Tipo de tarea: Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 01/10/2007	**Fecha fin:** 05/10/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se instala un paquete de software elegido por el usuario. El sistema detecta la lista de dependencias que es necesario procesar para instalar el paquete, conecta con el repositorio, descarga dichas dependencias y las instala en el orden requerido.	

Tarea de Ingeniería	
Número tarea: 2	**Número historia:**1
Nombre tarea: Actualizar paquete	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 15/10/2007	**Fecha fin:** 19/10/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se instala la versión superior del software que se desea actualizar y posteriormente si el proceso es exitoso se desinstala la versión antigua. Se debe tener en cuenta la detección de dependencias y verificar que en caso de un proceso de actualización en cascada afecte el funcionamiento de otros programas.	

Tarea de Ingeniería	
Número tarea: 3	**Número historia:** 1
Nombre tarea: Desinstalar paquete	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 08/10/2007	Fecha fin: 12/10/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se desinstala un paquete o programa de software elegido por el usuario.	

La historia #3 se divide en una sola tarea de ingeniería, la cual permitirá al sistema obtener la lista de paquetes disponibles en un repositorio, proceso que es necesario realizar antes de efectuar cualquier proceso de la gestión de paquetes. (Ver Figura 2.2)

Tarea de Ingeniería	
Número tarea: 4	**Número historia:** 3
Nombre tarea: Sincronizar con repositorio	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 20/10/2007	**Fecha fin:** 26/10/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: El sistema procede a conectar con el repositorio de Nova y descarga la última versión de la base de datos de paquetes disponibles para su instalación.	

Las pruebas de aceptación diseñadas por el cliente del sistema para verificar las funcionalidades están enfocadas mayormente en la primera historia de usuario, donde se comprobará la capacidad de la herramienta de instalar, actualizar y desinstalar software. Con respecto a la historia #3 se comprobará la correcta sincronización con el repositorio.

Pruebas de Aceptación	
Número del Caso de Prueba: 1	**Número de la Historia de Usuario:** #1
Nombre del Caso de Prueba: Instalación de un paquete	
Entradas: Se seleccionará un paquete de software a instalar.	
Resultado Esperado: El sistema detectará la lista de dependencias requeridas, conectará con el repositorio, descargará los paquetes y los procesará en el orden correcto.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes paquetes de software.	

Pruebas de Aceptación	
Número del Caso de Prueba: 2	**Número de la Historia de Usuario:** #1
Nombre del Caso de Prueba: Actualización de un paquete	
Entradas: Se seleccionará un paquete de software a actualizar	
Resultado Esperado: El sistema conectará con el repositorio, descargará la versión actualizada del paquete con sus correspondientes dependencias en caso de que existan y los instalará en el sistema. Posteriormente procederá a desinstalar la versión antigua del paquete.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes paquetes de software.	

Pruebas de Aceptación	
Número del Caso de Prueba: 3	**Número de la Historia de Usuario:** #1
Nombre del Caso de Prueba: Actualización de un paquete	
Entradas: Se seleccionará un paquete de software a desinstalar.	
Resultado Esperado: El sistema desinstalará el paquete seleccionado.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes paquetes de software.	

2.7.2.3. Iteración 2

En la segunda iteración se implementará una historia de usuario que no interactúa con el usuario por lo que no se le realizarán pruebas de aceptación. Dicha historia proveerá las funcionalidades que permitirán mostrar la información concerniente a los paquetes en pantalla de una forma sencilla de asimilar para el usuario de la herramienta. (Ver Figura 2.3)

Tarea de Ingeniería	
Número tarea: 8	**Número historia:** 4
Nombre tarea: Obtener paquetes instalados	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 05/11/2007	**Fecha fin:** 09/11/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se hace una consulta a la base de datos de los paquetes instalados para obtener una lista de los que podrán ser eliminados del sistema. Se debe incluir en la lista información de cada paquete con los campos: versión, categoría y descripción.	

Tarea de Ingeniería	
Número tarea: 9	**Número historia:** 4
Nombre tarea: Obtener paquetes disponibles	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 12/11/2007	**Fecha fin:** 16/11/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se hace una consulta a la base de datos de los paquetes disponibles para obtener una lista de los que podrán ser instalados en el sistema. Se debe incluir en la lista información de cada paquete con los campos: versión, categoría y descripción.	

Tarea de Ingeniería	
Número tarea: 10	**Número historia:** 4
Nombre tarea: Obtener paquetes actualizables	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 19/11/2007	**Fecha fin:** 23/11/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: A partir de la lista de paquetes instalados se itera por cada paquete y se compara con la última versión disponible en el repositorio (lista de paquetes disponibles). Si la versión instalada es menor, se marca como paquete actualizable.Se debe incluir en la lista información de cada paquete con los campos: versión, categoría y descripción.	

Tarea de Ingeniería	
Número tarea: 11	**Número historia:** 4
Nombre tarea: Diseño de la interfaz gráfica	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 26/112007	**Fecha fin:** 30/11/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se procede a diseñar una interfaz gráfica para la aplicación que debe constar de 3 vistas (actualizar, instalar y desinstalar) y una interfaz para la gestión de repositorios. La interfaz debe ser consistente con el entorno Gnome (*Human Interface Guidelines*). La interfaz debe mostrar por medio de un recurso visual el estado del paquete (instalado, actualizable o disponible) en la vista de paquetes a instalar.	

Tarea de Ingeniería	
Número tarea: 12	**Número historia:** 4
Nombre tarea: Mostrar información de paquetes	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 02/12/2007	**Fecha fin:** 07/12/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se muestra en la interfaz las listas de paquetes resultantes, con la correspondiente información.	

2.7.2.4. Iteración 3

En la tercera iteración se codificarán las historias de usuario restantes: #2 y #5.

La historia #2, permitirá configurar el comportamiento del gestor de paquetes del lado del cliente, permitiendo al usuario interactuar con los archivos de configuración del software. Se divide en tres tareas de ingeniería que proveen las funcionalidades de agregar, modificar y eliminar los repositorios sobre los cuales trabajará la herramienta. (Ver Figura 2.4)

Tarea de Ingeniería	
Número tarea: 5	**Número historia:** 2
Nombre tarea: Agregar repositorio	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 07/01/2008	**Fecha fin:** 11/01/2007
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se agrega la información de un repositorio al archivo de configuración de la herramienta.	

Tarea de Ingeniería	
Número tarea: 6	**Número historia:** 2
Nombre tarea: Eliminar repositorio	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 14/01/2008	**Fecha fin:** 18/01/2008
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se elimina la información de un repositorio del archivo de configuración de la herramienta.	

Tarea de Ingeniería	
Número tarea: 7	**Número historia:** 2
Nombre tarea: Modificar repositorio	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 21/01/2008	**Fecha fin: 25/01/2008**
Programador responsable: Daniel Hernandez Bahr	
Descripción: Se modifica la información de un repositorio previamente agregado al archivo de configuración de la herramienta.	

La historia #5 permite al usuario realizar búsquedas avanzadas en las diferentes vistas de la herramienta y solo está compuesta por una tarea de ingeniería que implementará la funcionalidad deseada. (Ver Figura 2.5)

Tarea de Ingeniería	
Número tarea: 13	**Número historia:**5
Nombre tarea: Buscar paquetes	
Tipo de tarea : Desarrollo Desarrollo / Corrección / Mejora / Otra (especificar)	
Fecha inicio: 21/01/2008	**Fecha fin: 25/01/2008**
Programador responsable: Daniel Hernandez Bahr	
Descripción: Realiza una busqueda en las diferentes vistas de paquetes, permitiendo además filtrar por categoria, nombre o descripción.	

En la presente iteración se le realizarán pruebas de aceptación diseñadas por el usuario al código generado por el proceso de implementación de cada una de las historias designadas. Ambas tienen una alta interacción con el usuario de la herramienta por lo que es necesario chequear la manera en que la herramienta cumple con las exigencias del cliente. Se chequeará la capacidad del software de agregar, modificar y eliminar repositorios de los archivos de configuración así como la habilidad de realizar búsquedas por nombre, categoría o descripción en las diferentes bases de datos de paquetes de software.

Pruebas de Aceptación	
Número del Caso de Prueba: 4	**Número de la Historia de Usuario:** #2
Nombre del Caso de Prueba: Agregar un repositorio	
Entradas: Se entrarán los datos del repositorio	
Resultado Esperado: El sistema agregará los datos correctamente al fichero de configuración.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes juegos de datos.	

Pruebas de Aceptación	
Número del Caso de Prueba: 5	**Número de la Historia de Usuario:** #2
Nombre del Caso de Prueba: Modificar un repositorio	
Entradas: Se entrarán los datos que se desean modificar de un repositorio	
Resultado Esperado: El sistema agregará los datos correctamente al fichero de configuración.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes juegos de datos.	

Pruebas de Aceptación	
Número del Caso de Prueba: 5	**Número de la Historia de Usuario:** #2
Nombre del Caso de Prueba: Eliminar un repositorio	
Entradas: Se seleccionará el repositorio que se desea eliminar.	
Resultado Esperado: El sistema eliminará los datos deseados del fichero de configuración.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes juegos de datos.	

Pruebas de Aceptación	
Número del Caso de Prueba: 6	**Número de la Historia de Usuario:** #5
Nombre del Caso de Prueba: Realizar búsqueda de paquetes	
Entradas: Se entrarán los datos del paquete que se desea buscar (palabras clave de la búsqueda), así como los campos en los cuales se realizará la búsqueda (nombre, categoría o descripción).	
Resultado Esperado: El sistema realizará la búsqueda en los campos deseados de la vista que se encuentra activa en la pantalla (Instalar, actualizar o desinstalar), devolviendo una lista de paquetes que satisfacen los parámetros de la búsqueda.	
Observaciones: Este caso de prueba debe realizarse varias veces, con diferentes juegos de datos, en cada una de las vistas de la herramienta.	

2.8. Conclusiones del Capítulo

Durante el transcurso del presente capítulo se realizó una descripción de la solución general propuesta, se definieron y redactaron las historias de usuario así como las tareas de la ingeniería correspondientes a cada una de ellas, estimándose un tiempo ideal de 35 días necesario para implementar la herramienta. Se definieron las pruebas de aceptación requeridas para garantizar que la implementación de cada una de las historias sea realizada correctamente y que el sistema se comporta tal y como se espera que lo haga. Además se generaron los diagramas UML considerados indispensables para asegurar la completa comprensión y claridad del sistema antes de comenzar a codificar las historias obtenidas.

3. Diseño del sistema

3.1. Introducción

El presente capítulo describe el proceso de diseño de la herramienta gestión de paquetes de software de Nova, a través de un conjunto de artefactos propuestos por XP para el desarrollo de software. Como exige la metodología, la aplicación está siendo codificada a medida que avanza investigación por lo que los diagramas y demás elementos de la documentación se encuentran en constante evolución y refinamiento por lo que solo se incluirán en éste trabajo las últimas versiones de cada uno de los artefactos. Se expondrán fundamentalmente las Tarjetas CRC y el modelado en lenguaje UML de la arquitectura del sistema.

3.2. Diseño de la Arquitectura del Sistema

Durante el ciclo de vida del desarrollo de un software utilizando XP la fase de diseño del software se realiza justo después que se termina la fase de planificación. Durante esta fase se comienza a definir el diseño arquitectónico del software partiendo de las historias de usuario de la primera iteración y de la metáfora, un elemento que según Kent Beck el creador de la metodología, guía al equipo de proyecto durante todo el desarrollo del programa y que además sustituye gran parte de lo que sería la arquitectura del sistema en modelos de desarrollo mas pesados o tradicionales.[5]

La arquitectura de la herramienta o de cualquier proyecto guiado por XP debe ser sencilla y responder al principio de "*cual es la solución más simple capaz de funcionar*", debe ser una estructura que organiza la lógica del sistema de manera que al realizar cambios en partes no provoca cambios en otros componentes a menos que sea absolutamente necesario. Debe enfocarse en como resolver los problemas identificados en el momento en que se diseña, sin tener en cuenta posibles variaciones o requerimientos futuros, los cuales serán enfrentados y agregados al diseño en el momento en que sean detectados.

3.2.1. Metáfora del sistema propuesto

La metáfora creada por Kent Beck es una historia simple de cómo funciona todo el sistema compartida por todo el equipo de desarrollo. Su misión es describir el software de manera que los desarrolladores y clientes que trabajan en conjunto puedan comunicarse fácilmente a la hora de referirse al sistema, o a una parte de éste. Cuando el sistema es pequeño la metáfora es extremadamente sencilla, pero cuando es un sistema complejo el que se desea describir, puede llevar una explicación que clarifique el funcionamiento del mismo.[6]

Teniendo en cuenta los parámetros anteriormente expuestos, la metáfora del sistema sería: Herramienta gráfica que permite instalar, desinstalar y actualizar paquetes de software desde un repositorio local o situado en Internet.

3.2.2. Arquitectura

Para lograr definir correctamente la arquitectura de un software es necesario identificar los elementos arquitectónicamente significativos (clases, módulos, objetos, interfaces) y las relaciones que existen entre ellos. Aunque el creador de la metodología XP sostiene que la arquitectura puede ser diseñada partiendo de las historias de usuario de la primera iteración y de la metáfora del proyecto para que sea refinada a medida que el ciclo de desarrollo avance, en la presente investigación se consideró más conveniente la aproximación de Martin Fowler a dicha situación. Fowler plantea que esa información generada en la primera iteración es insuficiente y propone una forma ágil de modelar una arquitectura apoyándose en artefactos generados en iteraciones siguientes, con el fin de obtener una mayor claridad del funcionamiento del sistema.[7] Para lograr esto se mantiene el precepto de “viajar ligero” y el culto a la simplicidad característicos de XP para basándose en ellos generar durante el ciclo de vida del proyecto solamente los diagramas y artefactos estrictamente necesarios para garantizar una total comprensión entre el equipo de desarrollo y los clientes.

Partiendo del principio de que la herramienta va a evolucionar y a desarrollarse activamente, es necesario garantizar que lo cambios o nuevas funcionalidades que serán añadidas puedan ser asimilados por la arquitectura del sistema. Para lograr esto se propone diseñar el programa modularmente de forma que para agregar un nuevo grupo de funcionalidades solo sea crear un módulo nuevo e integrarlo. Además la arquitectura de los módulos internamente y en general del programa estará formada por capas (presentación, lógica de negocio y acceso a datos) organizadas

por responsabilidades.

Tomando como punto de partida el enfoque de anteriormente expuesto, se propone para describir la arquitectura de la herramienta a desarrollar los diagramas de clases y diagramas de componentes, los cuales proveerán tanto una vista conceptual como física del funcionamiento del sistema.

3.3. Tarjetas CRC

Las tarjetas CRC son una técnica de modelado creadas para enfrentar los paradigmas de la programación orientada a objetos y ayudar a los desarrolladores de software a crear diseños de clases orientados a responsabilidades. Dichas tarjetas constan de tres secciones, nombre de la clase, responsabilidades y colaboradores. La sección responsabilidades se lista cada una de las funciones o tareas que debe ser capaz cumplir un objeto de dicha clase mientras que la sección colaboradores contiene otras clases del diseño que pueden colaborar para proveer datos o funcionalidades.
La metodología XP estipula en toda la bibliografía consultada el uso de las tarjetas CRC como un artefacto obligatorio durante el desarrollo de un proyecto debido a los beneficios que aporta en cuanto a la comunicación y transparencia del desarrollo. La mayoría de los autores recomiendan que se encuentren en un lugar público a la vista de todo el equipo donde todos los programadores puedan contribuir a mejorar y refinar el diseño resultante así como verificar que diferentes clases no dupliquen responsabilidades o que una interfiera con las funcionalidades de la otra; problemas muy frecuentes a la hora de desarrollar una arquitectura bajo paradigmas de orientación a objetos.

Las Tarjetas CRC de las clases significativas para significativas para la arquitectura del sistema pueden ser consultadas en el Anexo 3. Tarjetas CRC.

3.4. Diagrama de clases del Diseño

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.

En el caso propuesto se ha creado un diseño compuesto por capas, utilizando el Patrón Separación Modelo – Vista para desacoplar la interfaz gráfica de los modelos que implementan la lógica del

negocio y el acceso a datos; además se implementó un Patrón Fachada para crear una interfaz única mediante la cual las clases de interfaz se comunicarán con el resto del programa. (Ver Anexo 4. Diagrama de Clases del Diseño)

3.5. Diagrama de componentes

Un diagrama de componentes representa la separación de un sistema de *software* en componentes físicos y muestra las dependencias entre estos. Cada subsistema corresponde a un paquete físico y cada componente a un módulo, fichero o librería existente en la memoria de almacenado. (Ver Anexo 5. Diagrama de Componentes)

3.6. Propuesta de Interfaz Gráfica

La interfaz gráfica propuesta se diseñó utilizando el programa Glade y las librerías gráficas GTK. Contiene los elementos de diseño comunes encontrados en la mayoría de las herramientas de gestión de paquetes estudiadas y fue creada siguiendo los patrones de la Guía de Interfaz Humana del Proyecto Gnome, por lo que es visualmente consistente con el resto de las interfaces visuales de la distribución Nova y con el entorno de escritorio Gnome. Dicha consistencia visual garantiza que el usuario de la herramienta tenga un acceso rápido y sencillo a todas las funcionalidades que esta provee así como un entorno de trabajo agradable y cómodo. (Ver Anexo 6. Prototipo de Interfaz de Usuario)

3.7. Conclusiones del Capítulo

En el presente capitulo se abordó la fase de diseño de la herramienta, se obtuvo la metáfora que va a describir el sistema y se crearon las Tarjetas CRC de todas las clases que intervendrán en la solución con lo que se hizo una exploración de las relaciones y colaboraciones entre ellas, lo cual contribuyó a lograr un buen diseño. Se obtuvo una arquitectura robusta y flexible, capaz de asimilar variación de requisitos apoyada en la programación modular y un diseño en 3 capas lógicas descrito detalladamente mediante un Diagrama de Clases y un Diagrama de Componentes.

Se creó además un prototipo de interfaz gráfica que reúne las mejores características de las herramientas similares estudiadas y los mejores elementos de diseño presentes en la Guía de

Interfaz Humana de Gnome, lo que garantiza una completa aceptación por parte de los usuarios de la distribución Nova.

Conclusiones

Como resultado del estudio anterior ha sido posible la construcción de un sistema de gestión de paquetes que cumpla con los requerimientos básicos del sistema operativo NOVA.

1. La herramienta gráfica para efectuar la gestión de paquetes garantiza una mayor eficiencia y control sobre los procesos de instalación y desinstalación y actualización de software en el sistema operativo NOVA.
2. La herramienta gráfica creada optimiza el proceso de gestión de paquetes de software y permite que el usuario final pueda gestionar los paquetes en Nova con facilidad y sin necesidad de usar una consola.
3. Se logra que el usuario se sienta cómodo y sin esperas prolongadas instalando paquetes precompilados.
4. Se ejerce mayor control sobre la estabilidad del sistema operativo.
5. El uso de la metodología de programación extrema favoreció la elaboración de la herramienta gráfica.
6. El análisis de las ventajas y desventajas de los diferentes métodos permitió tomar lo mejor de ellos para concebir el prototipo de la herramienta con interfaz gráfica.

Recomendaciones

1. Continuar y profundizar el estudio del sistema de gestión de paquetes con el fin de mejorar progresivamente las funcionalidades del mismo.
2. Mejorar la funcionalidad de limpieza del sistema, incluyendo un algoritmo de eliminación de dependencias huérfanas, para disminuir el uso del medio de almacenamiento y la estabilidad del sistema.
3. Mejorar la integración con el sistema operativo y el entorno gráfico.
4. Extender su uso con otras bibliotecas gráficas (QT).

Referencias Bibliográficas

[1] Angel Goñi Oramas, “Proyecto Nova, base para crear un sistema operativo con repercusiones sociales,” 2007.

[2] ídem Referencia 1

[3] Angel Goñi Oramas, “Nova LNX como plataforma de desarrollo personalizada.,” *CD de Memorias Feria Internacional Informatica Habana*, 2007, pág. 10.

[4] Ídem Referencia 1

[5] Kent Beck, *Extreme Programming Explained*, Addison-Wesley, 2000.

[6] Ídem Referencia 5

[7] “Is Design Dead?”; http://martinfowler.com/articles/designDead.html.

Bibliografía

- “Adept - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Adept.
- “Advanced Packaging Tool - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Advanced_Packaging_Tool.
- “Ancho de banda - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Ancho_de_banda.
- “aptitude (program) - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Aptitude_%28program%29.
- “Archivo binario - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Archivo_binario.
- “Autopackage - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Autopackage.
- “Autopackage y klik, instalación fácil de software en GNU/Linux | Marble Station”; http://www.marblestation.com/blog/?p=529.
- “Biblioteca (programación) - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Biblioteca_%28programaci%C3%B3n%29.
- “Binding - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Binding.
- “Código fuente - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/C%C3%B3digo_fuente.
- “Conary - rPath Wiki”; http://wiki.rpath.com/wiki/Conary.
- “Cygwin - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Cygwin.
- “deb (file format) - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Deb_%28file_format%29.
- “Debian - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Debian.
- “Dependencias de software - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Dependencias_de_software.

- “Dependency hell - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Dependency_hell.
- “Desinstalación de software - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Desinstalaci%C3%B3n_de_software.
- “DistroWatch.com: Put the fun back into computing. Use Linux, BSD.”; http://distrowatch.com/weekly.php?issue=20050808.
- “DLL Hell - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/DLL_Hell.
- “Dpkg - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Dpkg.
- Kent Beck, *Extreme Programming Explained*, Addison-Wesley, 2000.
- “Fink - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Fink.
- “Gentoo Linux - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Gentoo_Linux.
- “GTK+ - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Gtk.
- “Instalación de software - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Instalaci%C3%B3n_de_software.
- “Interfaz gráfica de usuario - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario.
- “Librerías compartidas”; http://linux-cd.com.ar/manuales/freebsd/policies-shlib.html.
- “Linux distribution - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Linux_distribution.
- “Linux Standard Base - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Linux_Standard_Base.
- “Log (registro) - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Log_%28registro%29.
- “Mac OS - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Mac_OS.
- “MacPorts - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Darwinports.
- “Malware - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/C%C3%B3digo_maligno.

- ".NET - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/.NET.
- Angel Goñi Oramas, "Nova LNX como plataforma de desarrollo personalizada.," *CD de Memorias Feria Internacional Informatica Habana*, 2007, pág. 10.
- "Núcleo (informática) - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/N%C3%BAcleo_(inform%C3%A1tica).
- "Package management system - Wikipedia, the free encyclopedia"; http://en.wikipedia.org/wiki/Package_management_system#Meta_package_managers.
- "Pacman (Arch Linux) - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/Pacman_%28Arch_Linux%29.
- "Paquete de software - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/Paquete_de_software.
- "Pardus (operating system) - Wikipedia, the free encyclopedia"; http://en.wikipedia.org/wiki/PISI.
- "pkgsrc - Wikipedia, the free encyclopedia"; http://en.wikipedia.org/wiki/Pkgsrc.
- "Portage (software) - Wikipedia, the free encyclopedia"; http://en.wikipedia.org/wiki/Portage_%28software%29.
- "Ports collection - Wikipedia, the free encyclopedia"; http://en.wikipedia.org/wiki/Ports_collection.
- Angel Goñi Oramas, "Proyecto Nova, base para crear un sistema operativo con repercusiones sociales," 2007.
- "PyGTK - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/PyGTK.
- "Python - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/Python.
- "¿Qué es GNU/Linux?"; http://www.grulic.org.ar/linux.html.
- "RPM Package Manager - Wikipedia, la enciclopedia libre"; http://es.wikipedia.org/wiki/RPM_Package_Manager.
- Henrik Kniberg, *Scrum y XP desde las trincheras.*, C4Media Inc, 2007; http://infoq.com/minibooks/scrum-xp-fromthetrenches.

- “Sistema de gestión de paquetes - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Gestor_de_paquetes.
- “Sistema operativo - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Sistema_operativo.
- “Slackware - Wikipedia, the free encyclopedia”; file:///home/micha/.mozilla/firefox/x0g31f9w.default/ScrapBook/data/20080216162507/index.html.
- “Software package - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Software_package.
- “Software repository - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Linux_repository.
- “Sparkle: Free Software Updates for All!”; http://sparkle.andymatuschak.org/.
- “Synaptic Package Manager - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Synaptic_Package_Manager.
- “Synaptic - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Synaptic.
- Lisa Crispin y Tip House, *Testing Extreme Programming*, Addison-Wesley, 2002.
- “Tools/pirut - FedoraProject”; http://fedoraproject.org/wiki/Tools/pirut.
- “Unix - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Unix.
- “Urdiendo: Ubuntu-desktop: El metapaquete”; http://urdiendo.blogspot.com/2007/11/ubuntu-desktop-el-metapaquete.html.
- “Urpmi - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/Urpmi.
- “Windows Installer - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Windows_Installer.
- “YaST - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/YaST.
- “Yellow dog Updater, Modified - Wikipedia, the free encyclopedia”; http://en.wikipedia.org/wiki/Yellow_dog_Updater%2C_Modified.
- “Yum Extender - Yum Extender - New Yum Extender Development Site - News”;

http://www.yum-extender.org/cms/modules/news/.

- “YUM - Wikipedia, la enciclopedia libre”; http://es.wikipedia.org/wiki/YUM.

Anexos

Anexo 1. Modelo del Dominio

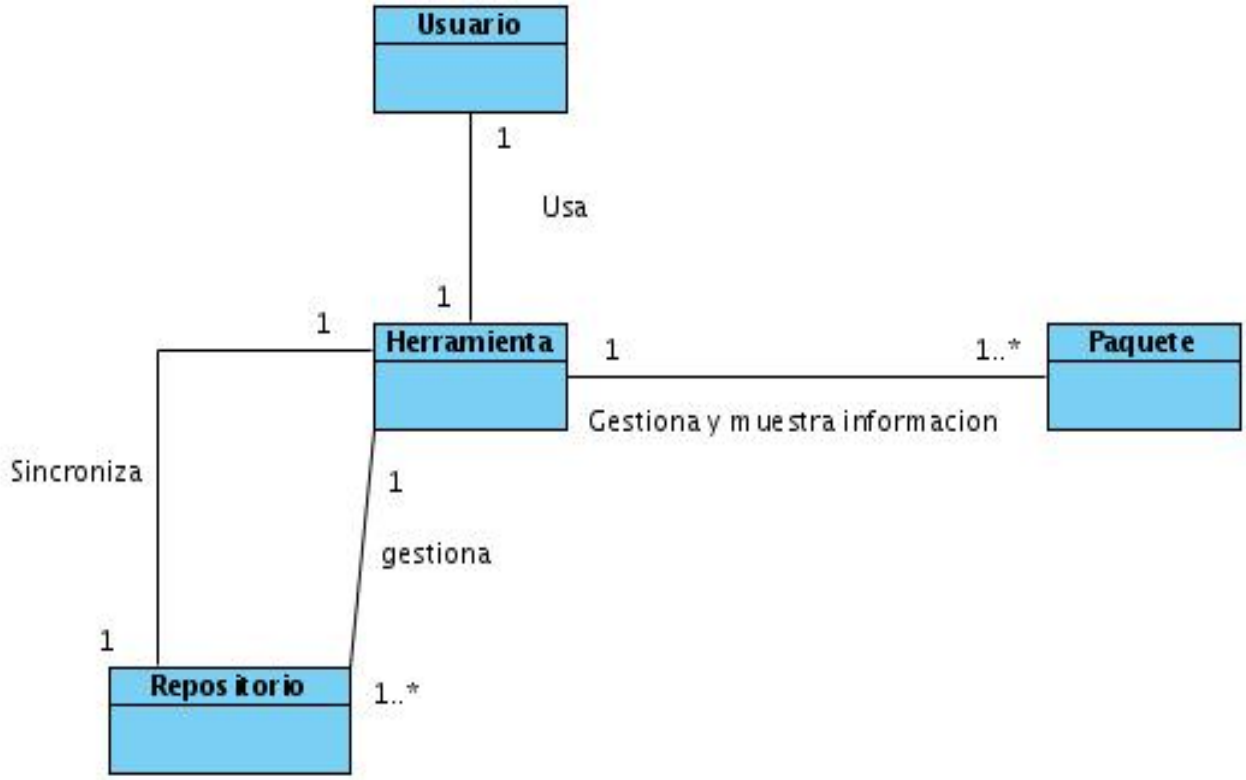


Ilustración I: Modelo del Dominio.

Anexo 2. Diagramas de Secuencia

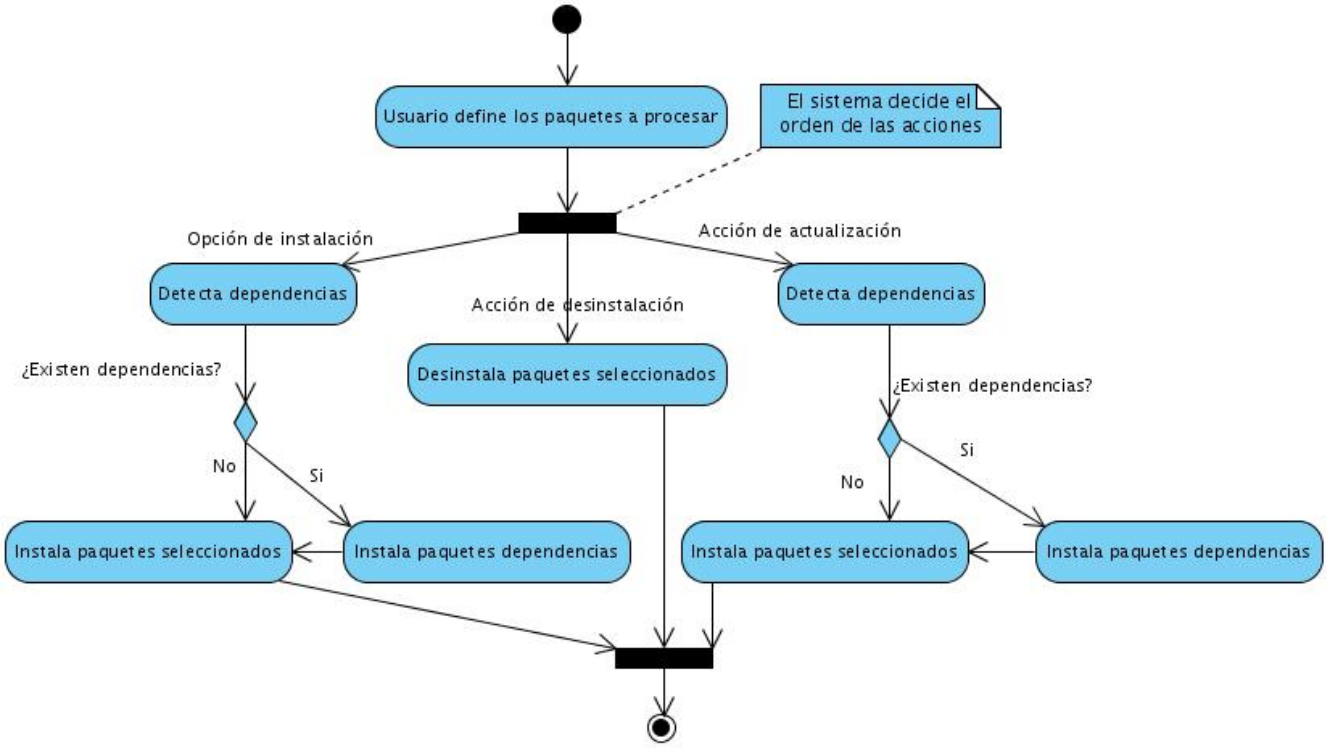


Ilustración II: Digrama de secuencia Gestionar Paquetes.

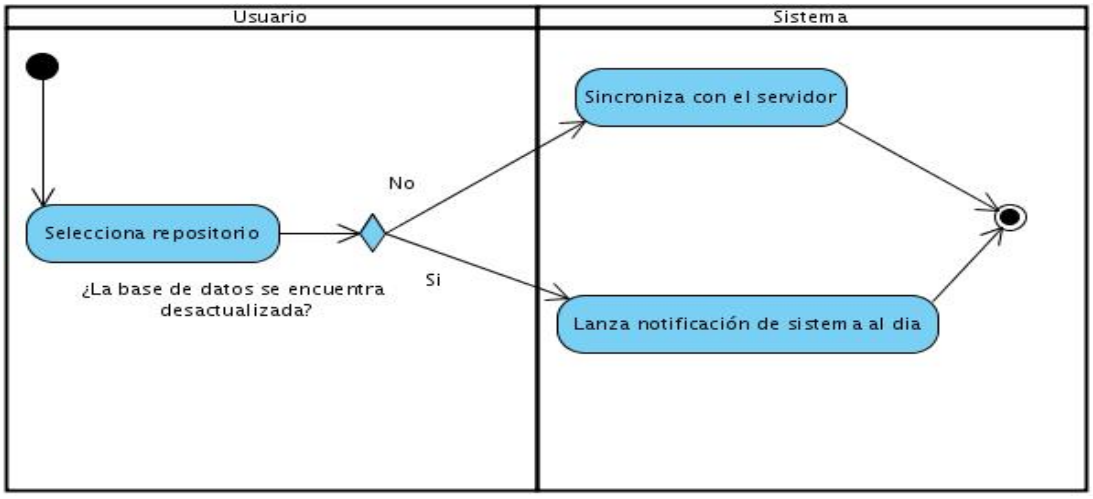


Ilustración III: Diagrama de secuencia. Sincronizar con Repositorio.

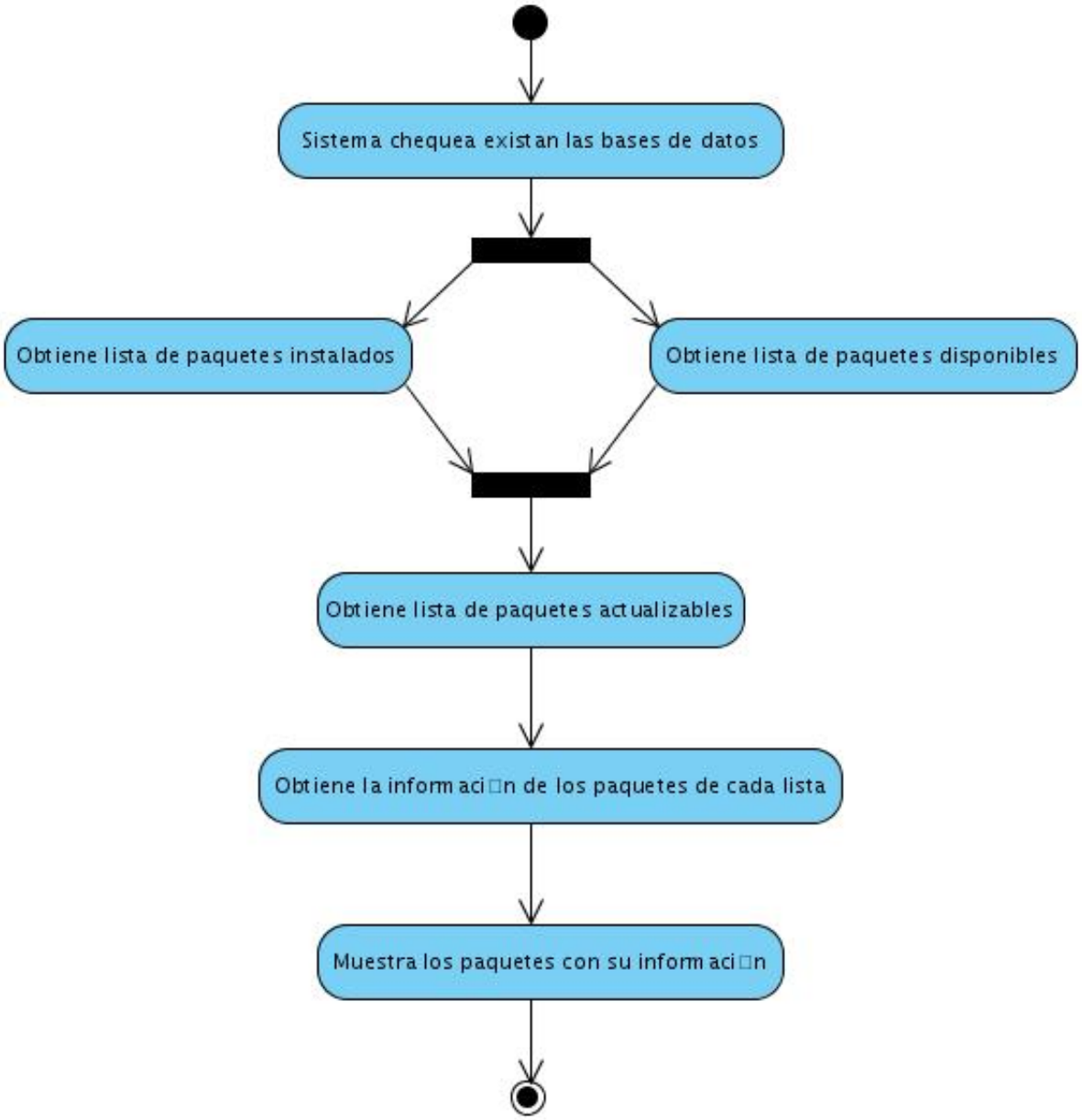


Ilustración IV: Diagrama de secuencia. Mostrar Paquetes.

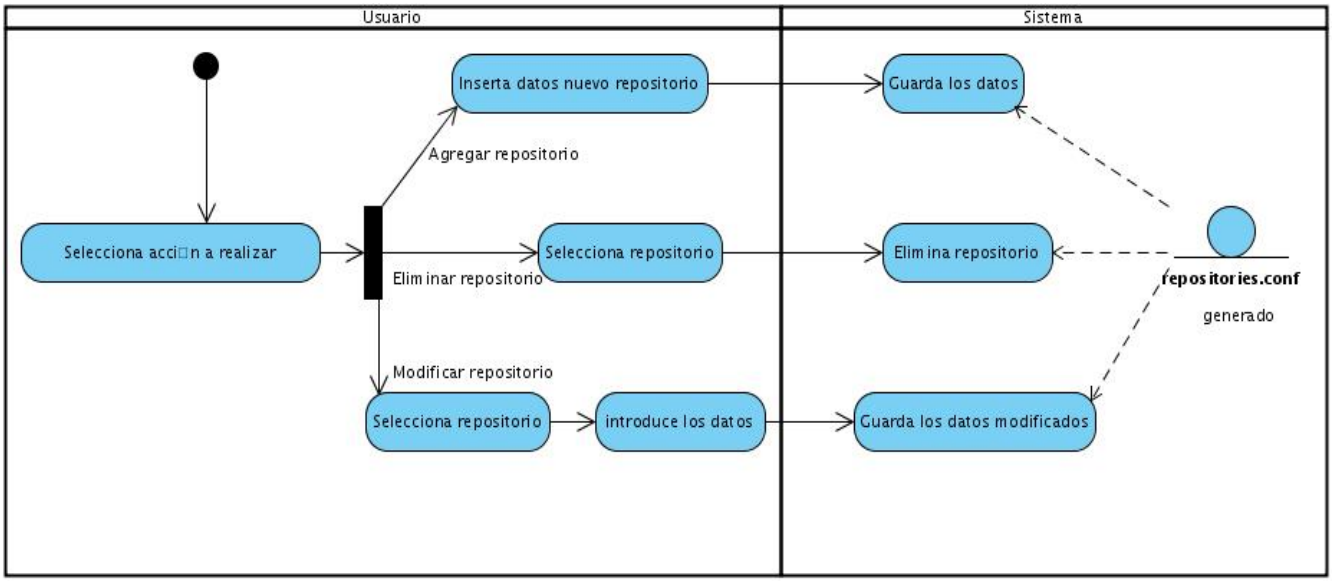


Ilustración V: Diagrama de Secuencia. Gestionar Repositorios.

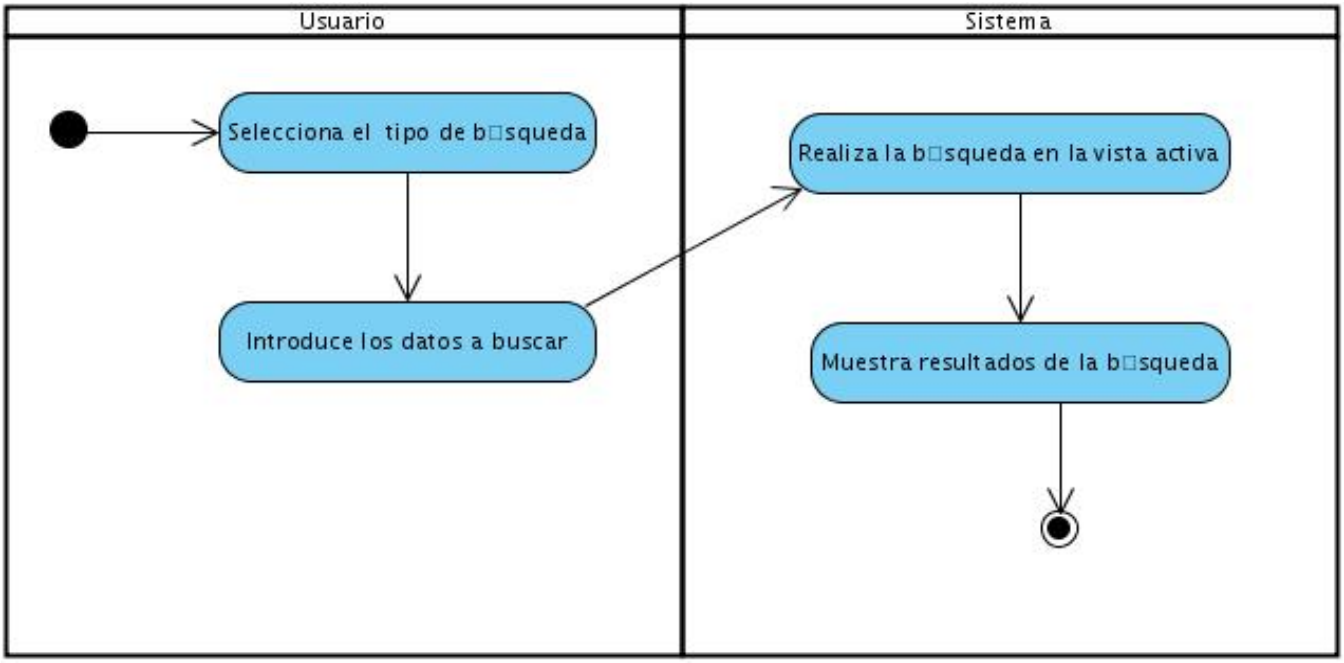


Ilustración VI: Diagrama de Secuencia. Buscar Paquete.

Anexo 3. Tarjetas CRC

Equo	
SuperClasses: EquoInterface	
Descripción: Clase de control para el proceso de gestión de paquetes	
Atributos:	
Nombre	Descripción
progress_bar	Barra de progreso para el proceso de gestión de paquetes
progress_log	Log para el proceso de gestión de paquetes
Responsabilidades:	
Nombre	Colaborador
Conectar el proceso de gestión con la interfaz gráfica (connect_to_gui)	DummyBar DummyLog GuiUrlFetcher Package
Actualizar el progreso del proceso de gestión (update_progress)	DummyBar DummyLog GuiUrlFetcher Package

GuiUrlFetcher	
SuperClasses: UrlFetcher	
Descripción: Clase utilizada para conectar el proceso de descarga de los paquetes con la barra de progreso	
Atributos:	
Nombre	Descripción
progress_bar	Barra de progreso para el proceso de gestión de paquetes
Responsabilidades:	
Nombre	Colaborador
Conectar a la interfaz gráfica (connect_to_gui)	
Actualizar el progreso de la barra (update_Progress)	

Package	
SuperClasses: PackageInterface	
Descripción: Clase para ejecutar los pasos de gestión de un paquete	
Atributos:	
Nombre	Descripción
Responsabilidades:	
Nombre	Colaborador
Obtener los mensajes de Summon (summon_messages)	
Ejecutar los pasos de gestión de un paquete (run_stepper)	

PackageDataAccess	
Descripción: Clase de acceso a las bases de datos de paquetes instalados y disponibles	
Atributos:	
Nombre	Descripción
Responsabilidades:	
Nombre	Colaborador
Obtener paquetes disponibles	Equo
Obtener paquetes instalados	Equo
Obtener información de un paquete	Equo

Processor	
Descripción: Clase para realizar el proceso de gestión de paquetes.	
Atributos:	
Nombre	Descripción
Responsabilidades:	
Nombre	Colaborador
Detectar dependencia para la instalación (get_install_dependencies)	Equo
Detectar dependencia para la desinstalación (get_remove_dependencies)	Equo
Descargar paquetes necesarios para el proceso (fetch_packages)	Equo
Instalar paquetes (install_packages)	Equo
Desinstalar paquetes (remove_packages)	Equo

PackageInterface	
Descripción: Clase controladora que va a manejar las listas de paquetes extraídas de las Bases de datos y convertirlos en modelos a utilizar por GTK.	
Atributos:	
Nombre	Descripción
update_model	Árbol de paquetes actualizables
install_model	Árbol de paquetes disponibles
remove_model	Árbol de paquetes desinstalables
Responsabilidades:	
Nombre	Colaborador
Cargar y salva modelo de paquetes disponibles (load_imodel, save_imodel)	SummonConf PackageDataAccess
Cargar y salva modelo de paquetes actualizables (load_umodel, save_umodel)	SummonConf PackageDataAccess
Cargar y salva modelo de paquetes desinstalables (load_rmodel, save_rmodel)	SummonConf PackageDataAccess
Modificar el estado de un paquete en los modelos (update_package_status)	

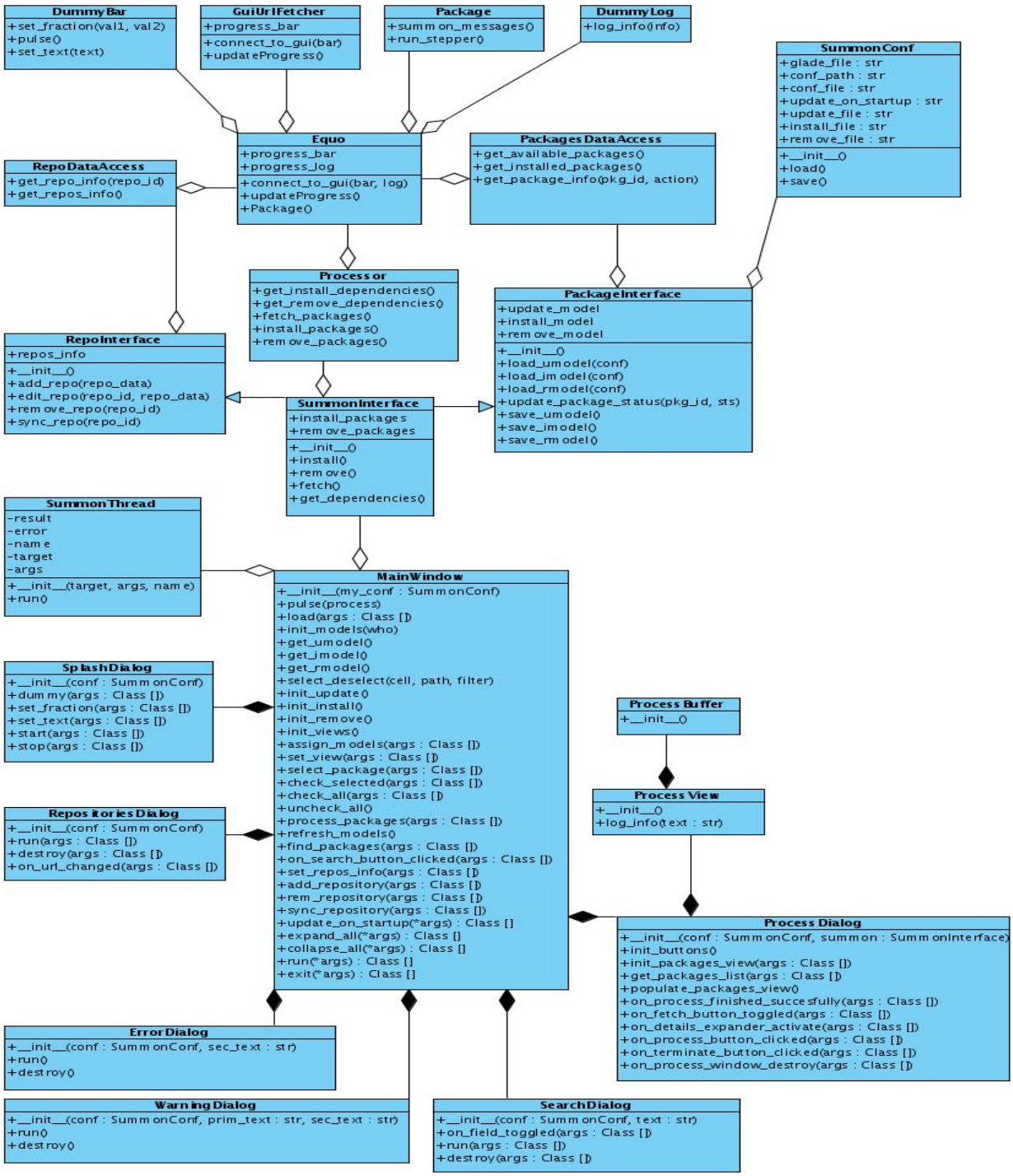
RepoInterface	
SubClasses: SummonInterface	
Descripción: Clase manejadora que procesa los datos de los repositorios antes de escribirlos a los ficheros de configuración o mostrarlos en pantalla	
Atributos:	
Nombre	Descripción
repos_info	Información de los repositorios
Responsabilidades:	
Nombre	Colaborador
Añadir modificar o eliminar un repositorio (add_repo, edit_repo, remove_repo)	RepoDataAccess
Sincronizar con el árbol de paquetes del repositorio (sync_repo)	

SummonConf	
Descripción: Clase de acceso al archivo de configuración de summon.	
Atributos:	
Nombre	Descripción
glade_file	Nombre del fichero de la interfaz
conf_path	Dirección del archivo de configuración
conf_file	Nombre del fichero de configuración
update_on_startup	Opción de actualizar al inicio del programa
update_file	Nombre del fichero lista de paquetes actualizables
install_file	Nombre del fichero lista de paquetes disponibles
remove_file	Nombre del fichero lista de paquetes desinstalables
Responsabilidades:	
Nombre	Colaborador
Cargar configuración (load)	
Guardar Configuración (save)	

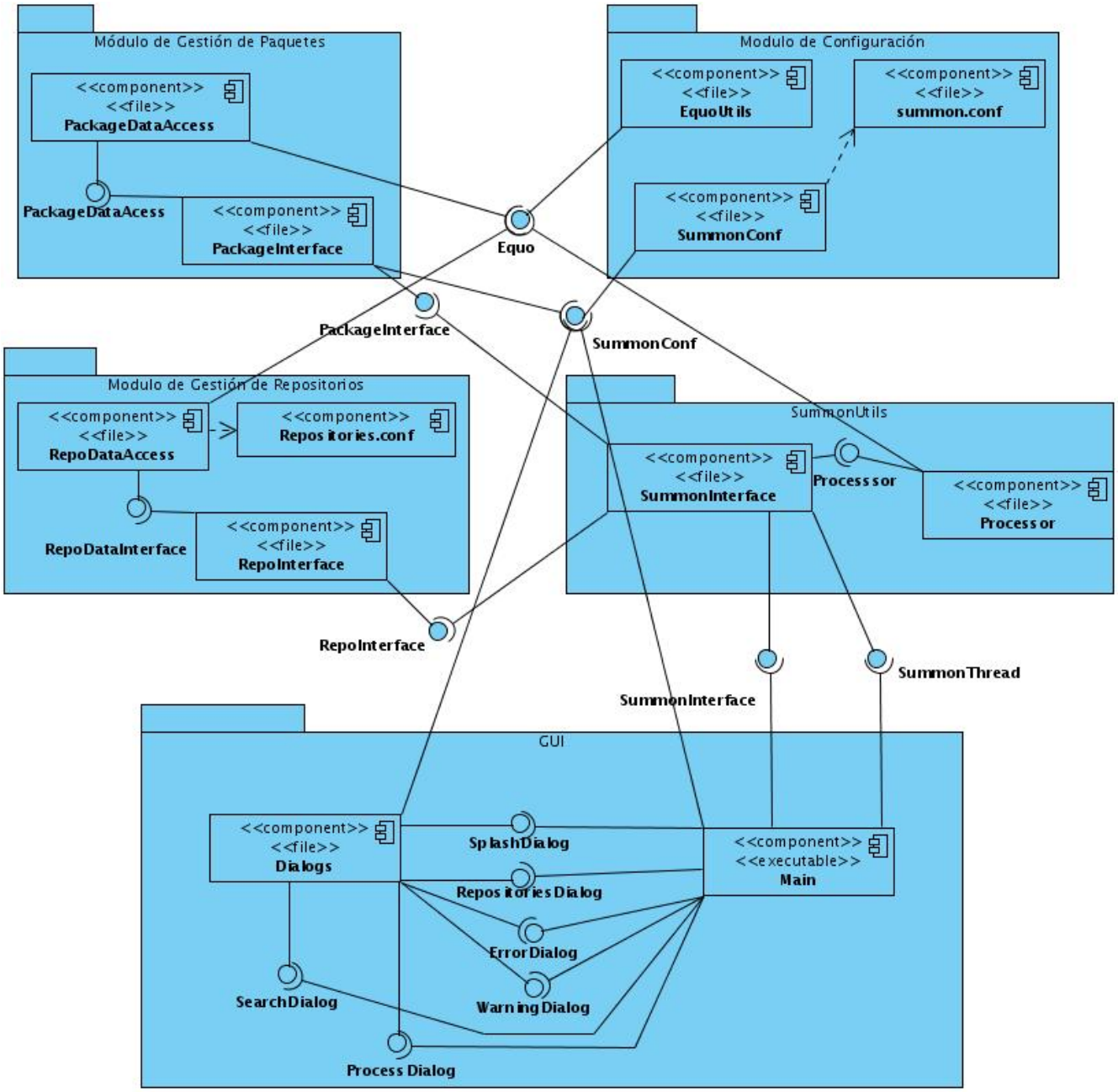
SummonInterface	
SuperClasses: RepoInterface PachageInterface	
Descripción: Clase fachada para acceder a las funcionalidades del programa. Provee la interfaz principal para acceder a la capa de la lógica del negocio.	
Atributos:	
Nombre	Descripción
install_packages	Lista de paquetes a instalar
remove_packages	Lista de paquetes a desinstalar
Responsabilidades:	
Nombre	Colaborador
Interfaz para instalar los paquetes seleccionados(install)	Processor
Interfaz para desinstalar los paquetes seleccionados(remove)	Processor
Interfaz para descargar los paquetes seleccionados(fetch)	Processor
Interfaz para obtener el listado de dependencias a satisfacer para efectuar el proceso de gestión (get_dependecies)	Processor

RepoDataAccess	
Descripción: Clase de acceso a los ficheros de configuración de los repositorios.	
Atributos:	
Nombre	Descripción
Responsabilidades:	
Nombre	Colaborador
Obtener información de un repositorio	Equo
Obtener información de todos los repositorios	Equo

Anexo 4. Diagrama de Clases del Diseño



Anexo 5. Diagrama de Componentes



Anexo 6. Prototipo de Interfaz de Usuario

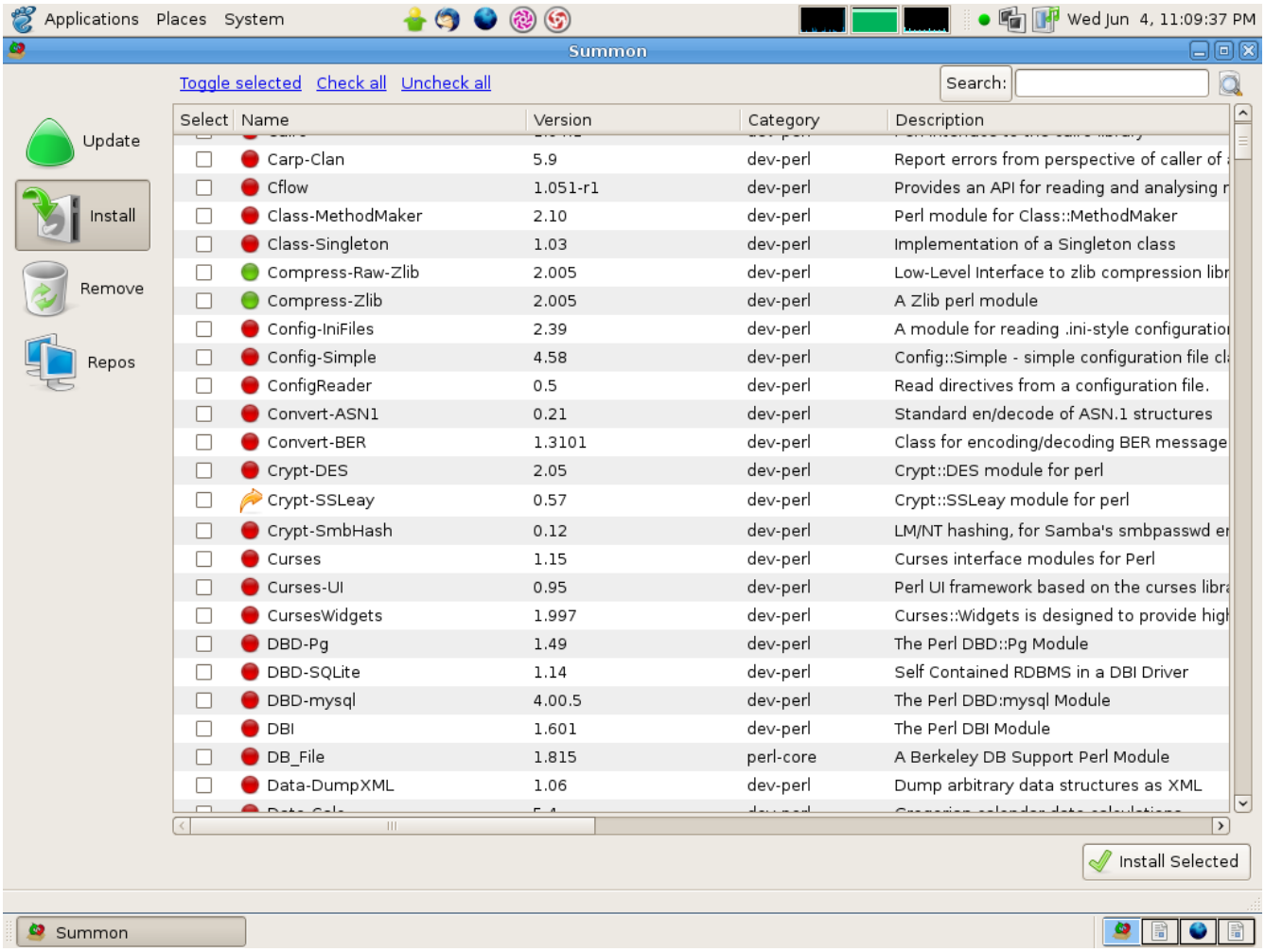


Ilustración VII: Vista de instalación

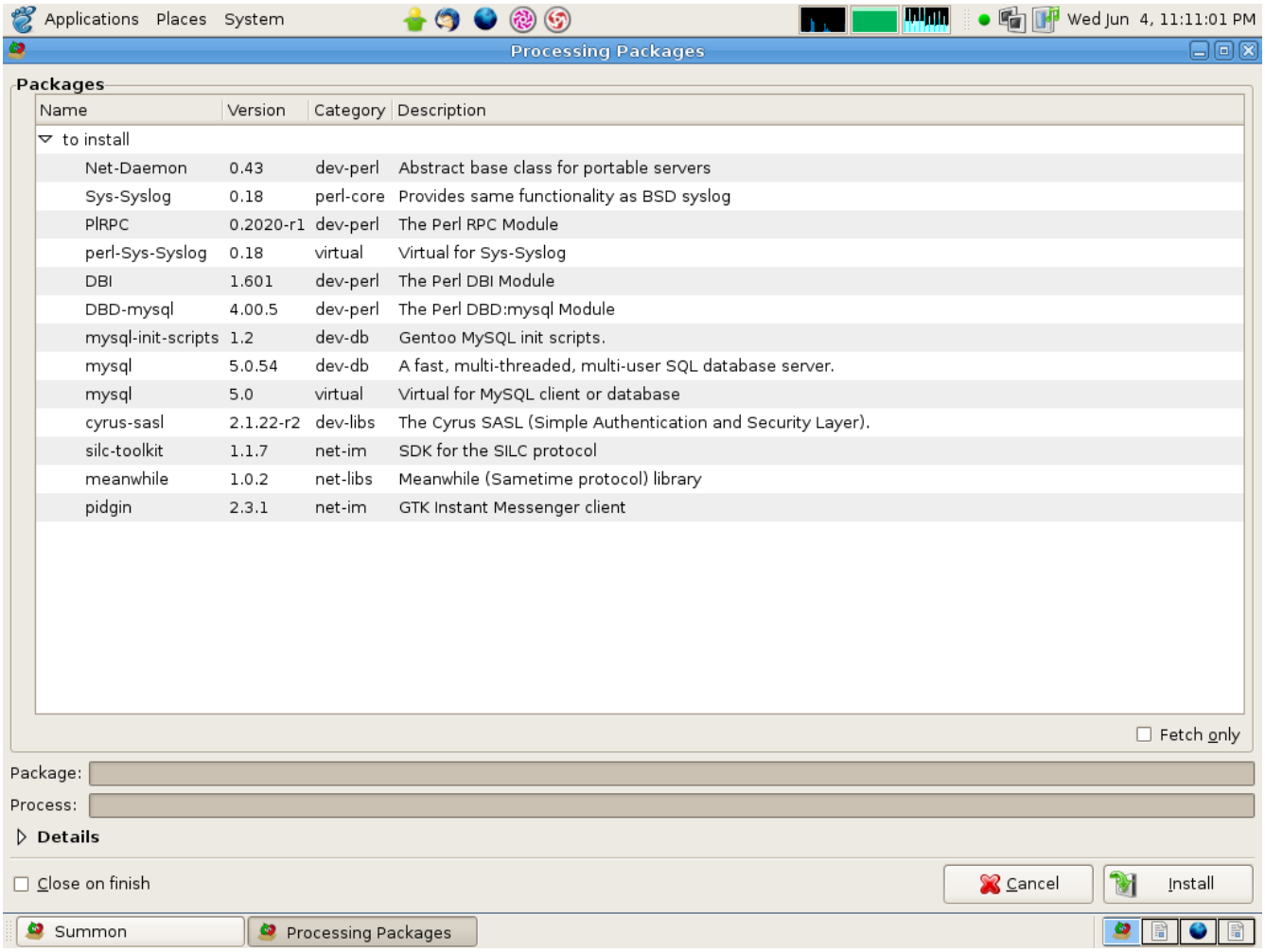


Ilustración VIII: Vista del procesamiento de un conjunto de paquetes.

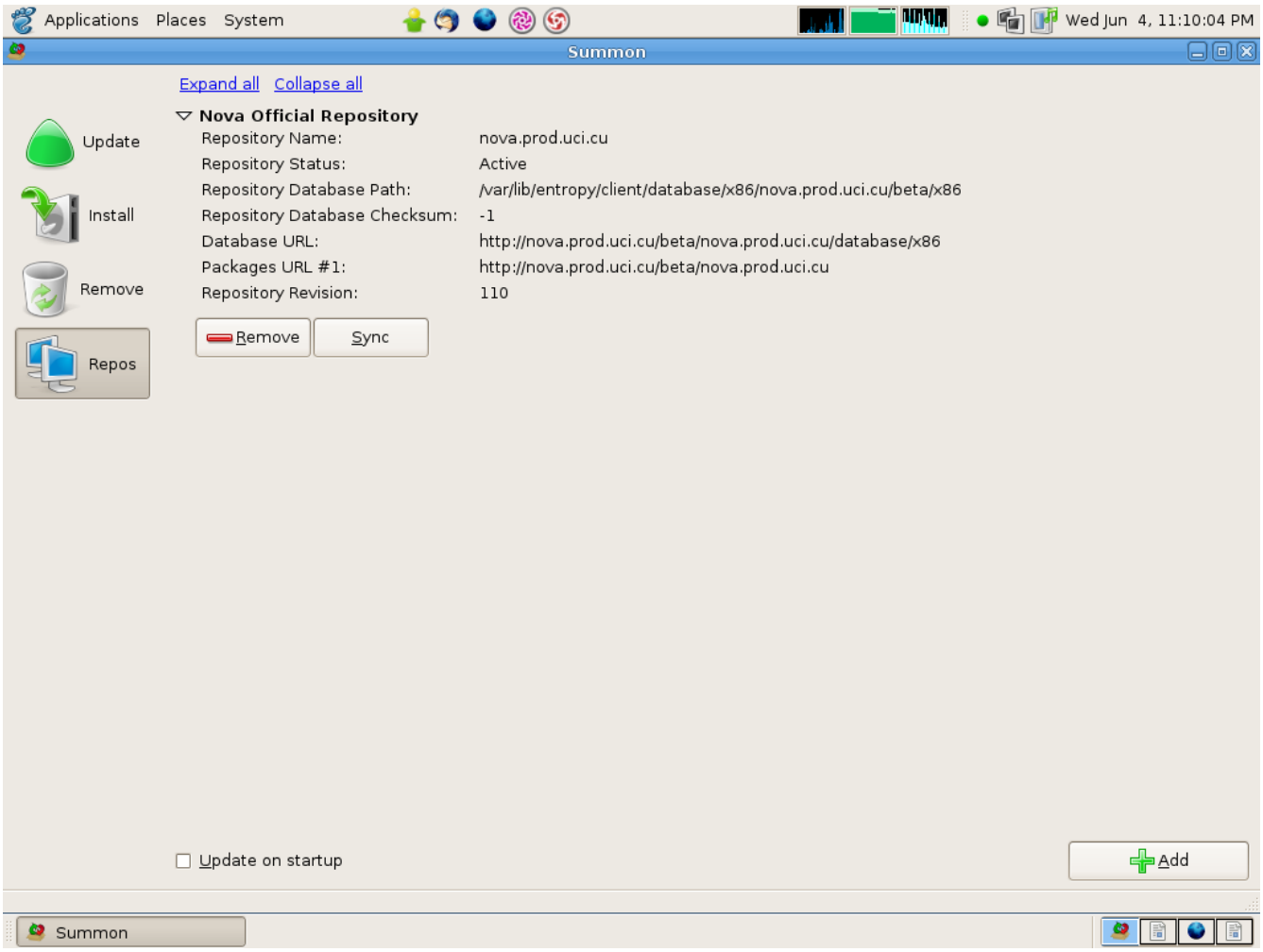


Ilustración IX: Vista de configuración de los repositorios.

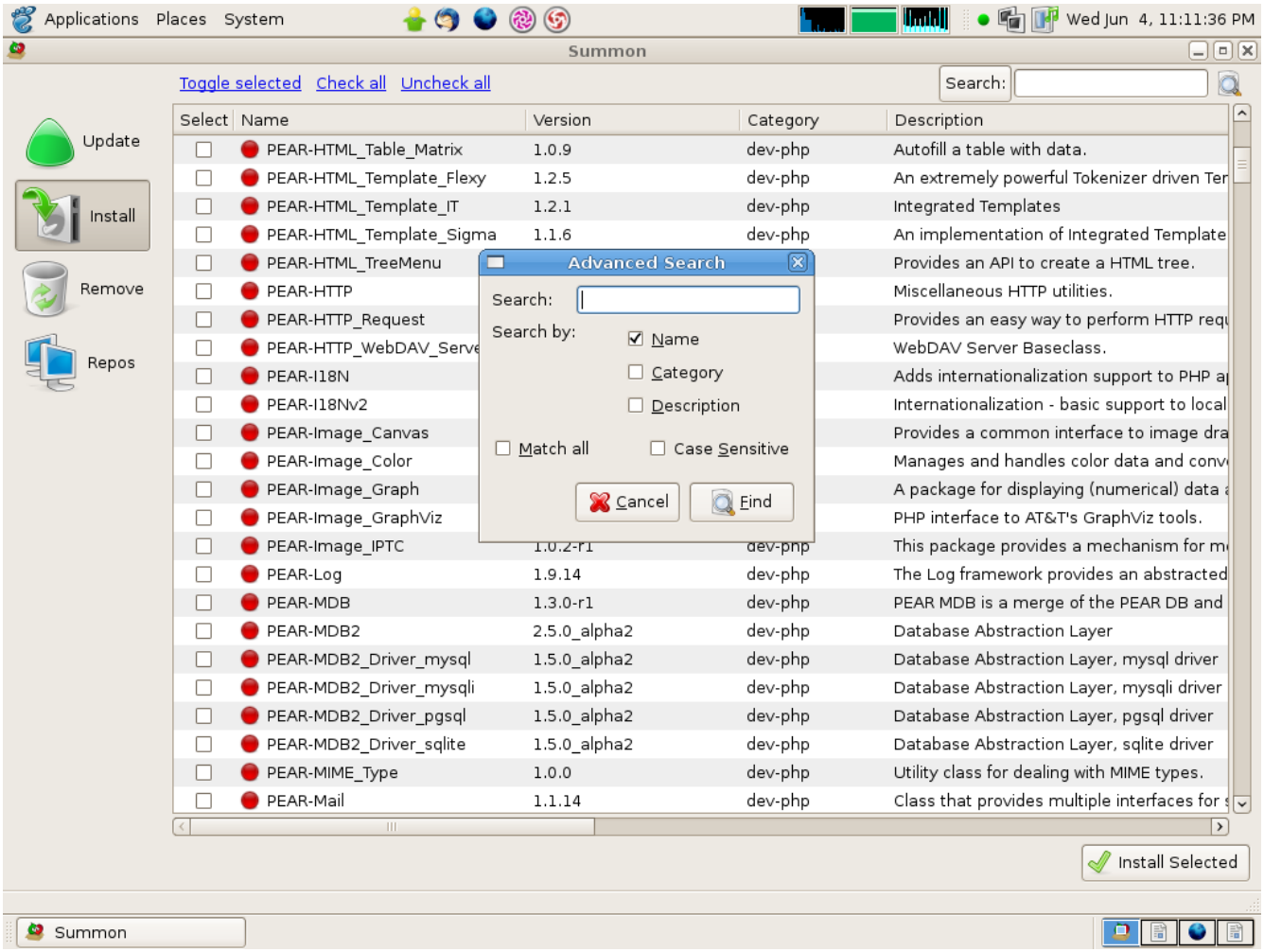


Ilustración X: Vista para la búsqueda avanzada de paquetes.

Anexo 7. Glosario de Términos

Ancho de banda: Cantidad de datos que se pueden transmitir en una unidad de tiempo. Por ejemplo, una línea ADSL de 256 kbps puede, teóricamente, enviar 256000 bits por segundo. Esto es en realidad la tasa de transferencia máxima permitida por el sistema, que depende del ancho de banda analógico, de la potencia de la señal, de la potencia de ruido y de la codificación de canal.

Archivos binarios: Archivo informático que contiene información de cualquier tipo, codificada en forma binaria para el propósito de almacenamiento y procesamiento en ordenadores. Un archivo binario que sólo contiene información de tipo textual sin información sobre el formato del mismo se dice que es un archivo de texto plano. En el caso particular de este documento se refiere a aquellos archivos binarios que son ejecutados en el procesador.

Biblioteca: Término informático para referirse a las bibliotecas de vínculos dinámicos, conocidas también como librerías.

Bug: Un defecto de software, es el resultado de un fallo o deficiencia durante el proceso de creación de programas de ordenador o computadora. Dicho fallo puede presentarse en cualquiera de las etapas del ciclo de vida del software aunque los más evidentes se dan en la etapa de desarrollo y programación. Los errores pueden suceder en cualquier etapa de la creación de software.

Ciclo de liberación: Cada distribución de GNU/Linux posee un ciclo de liberación que no es más que la frecuencia con que sale una nueva versión de la misma y que puede estar dada en semanas, meses o años.

Código fuente: Puede definirse como:

- Un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos.
- Un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de maquina mediante compiladores, ensambladores o intérpretes.

Normalmente está destinado a ser traducido a otro código, llamado código objeto, ya sea lenguaje máquina nativo para ser ejecutado por una computadora o código byte para ser ejecutado por un intérprete.

Consola: Programa que tiene como objetivo principal, la interacción del usuario con el sistema

operativo a base de líneas de comandos. Las consolas más usadas por los usuarios son las consolas virtuales, que son las que se pueden ver desde su entorno gráfico, por ejemplo: Gnome-terminal, Kterm y Yaquake.

Consumo de recursos: Término informático que se refiere a la forma en que el software interactúa con los recursos de hardware de un ordenador.

Directorio raíz: Ver la definición de Root.

Desinstalar: Proceso de desinstalación de software donde se elimina el software no deseado o innecesario y se revierten los cambios producidos en un sistema.

Dependencias huérfanas: Término informático que se refiere a aquellas dependencias que ya no cumplen objetivo en el sistema operativo debido a una ineficaz gestión de software.

Dependencia de software: En el campo del software una dependencia es una aplicación o una biblioteca requerida por otro programa para poder funcionar correctamente. Por ello se dice que dicho programa depende de tal aplicación o biblioteca.

DLL: Dynamic Link Library ("Biblioteca de vínculos dinámicos") es un archivo que contiene funciones que se pueden llamar desde aplicaciones u otras DLL, consisten en un conjunto de código común que puede estar compartido entre varias aplicaciones. Los desarrolladores utilizan las DLL para poder reciclar el código y aislar las diferentes tareas.

Gentoo: Gentoo Linux es una distribución GNU/Linux orientada a usuarios con cierta experiencia en este sistema operativo, fue fundada por Daniel Robbins, basada en la inactiva distribución llamada Enoch Linux.

GNU/Linux: GNU/Linux es, a simple vista, un Sistema Operativo. Es una implementación de libre distribución UNIX para computadoras personales, servidores, y estaciones de trabajo. Fue desarrollado para el i386 y ahora soporta los procesadores i486, Pentium, Pentium Pro y Pentium II, así como los clones AMD y Cyrix. También soporta máquinas basadas en SPARC, DEC Alpha, PowerPC/PowerMac, y Mac/Amiga Motorola 680x0.

Como sistema operativo, GNU/Linux es muy eficiente y tiene un excelente diseño. Es multitarea, multiusuario, multiplataforma y multiprocesador; en las plataformas Intel corre en modo protegido; protege la memoria para que un programa no pueda hacer caer al resto del sistema; carga sólo las partes de un programa que se usan; comparte la memoria entre programas aumentando la velocidad y disminuyendo el uso de memoria; usa un sistema de memoria virtual por páginas; utiliza toda la

memoria libre para cache; permite usar bibliotecas enlazadas tanto estática como dinámicamente; se distribuye con código fuente; usa hasta 64 consolas virtuales; tiene un sistema de archivos avanzado pero puede usar los de los otros sistemas; y soporta redes tanto en TCP/IP como en otros protocolos.

Interfaces (API): API (del inglés Application Programming Interface - Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Interfaz gráfica (GUI): En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario, es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

Instalación: Término informático que se refiere a la actualización del sistema operativo con nuevos paquetes que pueden ser de software, documentación, código fuente, bibliotecas, etc.

IP: El Protocolo de Internet (del inglés Internet Protocol) es un protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

Kernel: Ver definición de Núcleo.

Licencia libre: Copyleft o copia permitida comprende a un grupo de derechos de propiedad intelectual caracterizados por eliminar las restricciones de distribución o modificación de las que adolece el copyright, con la condición de que el trabajo derivado se mantenga con el mismo régimen de propiedad intelectual que el original.

Pueden protegerse una gran diversidad de obras, tales como programas informáticos, arte, cultura y ciencia, es decir prácticamente casi cualquier tipo de producción creativa.

Sus partidarios la proponen como alternativa a las restricciones que imponen las normas planteadas en los derechos de autor, a la hora de hacer, modificar y distribuir copias de una obra determinada. Se pretende garantizar así una mayor libertad para que cada receptor de una copia, o una versión derivada de un trabajo, pueda, a su vez, usar, modificar y redistribuir tanto el propio trabajo como las versiones derivadas del mismo. Así, y en un entorno no legal, puede considerarse como opuesto al copyright o derechos de autor tradicionales.

LINUX: Núcleo presente en los sistemas GNU/Linux.

Linux Standard Base: La Base Estándar para Linux (Linux Standard Base, abreviado LSB), es un proyecto conjunto de varias Distribuciones de Linux bajo la estructura organizativa del Free Standards

Group con el objeto de crear y normalizar la estructura interna de los sistemas operativos derivados de Linux. La LSB está basada en la Especificación POSIX, la Especificación Única de UNIX (Single UNIX Specification) y en varios otros estándares abiertos, aunque extiende estos en ciertas áreas.

Librería: Ver la definición de biblioteca.

Log: Un registro oficial de eventos durante un periodo de tiempo en particular. Para los profesionales en seguridad informática un log es usado para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo en particular o aplicación.

La mayoría de los logs son almacenados o desplegados en el formato estándar, el cual es un conjunto de caracteres para dispositivos comunes y aplicaciones. De esta forma cada log generado por un dispositivo en particular puede ser leído y desplegado en otro diferente.

También se le considera cómo aquel mensaje que genera el programador de un sistema operativo, alguna aplicación o algún proceso, en virtud del cual se muestra un evento del sistema.

Mac OS: Macintosh Operating System (Sistema Operativo de Macintosh), es el nombre del primer sistema operativo de Apple para los ordenadores Macintosh. El Mac OS original fue el primer sistema operativo con una interfaz gráfica de usuario en tener éxito.

Metapaquete: Un metapaquete es un como un paquete que enlaza varios paquetes, es decir, es un paquete que contiene referencias a varios paquetes pero no realiza ninguna funcionalidad.

MFC: Microsoft Foundation Class Library es una biblioteca que envuelve porciones de la API de Windows en clases de C++, incluyendo la funcionalidad que las habilita para usar la plataforma de la aplicación por defecto.

.NET: Es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Basado en ella, la empresa intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el sistema operativo hasta las herramientas de mercado.

Núcleo: En informática, el núcleo (también conocido en español con el anglicismo kernel) es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema. Como hay muchos programas y el acceso al hardware es limitado, el núcleo también se encarga de decidir qué programa podrá hacer uso de un

dispositivo de hardware y durante cuánto tiempo, lo que se conoce como multiplexado. Acceder al hardware directamente puede ser realmente complejo, por lo que los núcleos suelen implementar una serie de abstracciones del hardware. Esto permite esconder la complejidad, y proporciona una interfaz limpia y uniforme al hardware subyacente, lo que facilita su uso para el programador.

Paquete de software: Un paquete de software es una serie de programas que se distribuyen conjuntamente. Algunas de las razones para ello suelen ser que el funcionamiento de cada uno complementa o requiere a los demás, que sus objetivos están relacionados o como estrategia de mercadotecnia.

Muchos sistemas operativos modernos emplean sistemas de control de paquetes que permiten que el administrador del sistema instale o desinstale estos, sin que en ningún momento queden programas instalados que no funcionen por falta de otros incluidos en su paquete. El sistema de control de paquetes usualmente también se ocupa de mantener las dependencias entre paquetes: Si algún paquete depende de otro, el sistema se encarga de instalar este primero.

Paquete precompilado: Término informático para referirse al paquete de software en los sistemas GNU/Linux, con la diferencia de que generalmente estos paquetes son compilados por la organización que lo distribuye, por ejemplo, NOVA, Fedora, Mandriva, etc.

Programa malicioso: Malware malicious software, también (del inglés badware) es un software que tiene como objetivo infiltrarse en o dañar un ordenador sin el conocimiento de su dueño y con finalidades muy diversas.

Esta expresión es un término general muy utilizado por profesionales de la computación para definir una variedad de software o programas de códigos hostiles e intrusivos. Muchos usuarios de computadores no están aún familiarizados con este término y otros incluso nunca lo han utilizado. Sin embargo la expresión "virus informático" es más utilizada en el lenguaje cotidiano y a menudo en los medios de comunicación para describir todos los tipos de malware.

Rendimiento: Un término usado en la informática para dar medida de cuán eficaz puede ser un software en el uso de los recursos de harware.

Repositorio de Paquetes de Software: En algunas ocasiones conocido como “repo” es un lugar donde se almacenan paquetes de software de forma centralizada que pueden ser descargados para luego ser instalados en el sistema. Los mismos son descargados desde los servidores de internet generalmente a través de los protocolos HTTP, FTP y HTTPS.

Los repositorios de paquetes de software están preparados para distribuirse habitualmente sirviéndose de una red informática como Internet o en un medio físico como un disco compacto. Y pueden ser de acceso público, o pueden estar protegidos y necesitar de una autentificación previa.

A diferencia de los ordenadores personales o de las PC de escritorio, dichos depósitos suelen contar con sistemas de Backup y mantenimiento preventivo y correctivo, lo que hace que la información se pueda recuperar en el caso que nuestra máquina o PC quede inutilizable.

Root: El término informático “root” (raíz), en la familia de sistemas Unix/Linux/BSD, suele tener doble significado:

1. Directorio raíz “/” del cual se derivan los demás directorios del sistema, es decir, es el directorio padre. Comúnmente los directorios que se desprenden del root son “bin, boot, dev, etc, proc, home, mnt, sbin, sys, root, usr, tmp, var...”
2. Súper usuario del sistema, es decir, este tipo de usuarios tiene el permiso de hacer cualquier cosa con el sistema, ser root requiere un alto grado de responsabilidad.

Sistema de gestión de paquetes: Un sistema de gestión de paquetes, también conocido como gestor de paquetes, es una colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software. El término se usa comúnmente para referirse a los gestores de paquetes en sistemas UNIX.

En estos sistemas, el software se distribuye en forma de paquetes, frecuentemente encapsulado en un solo archivo. Estos paquetes incluyen otra información importante, además del software mismo, como pueden ser el nombre completo, una descripción de su funcionalidad, el número de versión, el distribuidor del software, la suma de verificación y una lista de otros paquetes requeridos para el correcto funcionamiento del software. Esta metainformación se introduce normalmente en una base de datos de paquetes local.

En los sistemas donde las aplicaciones comparten trozos de instrucciones, como en la mayor parte de las distribuciones de Linux, la resolución de dependencias al gestionar software se convierte en una necesidad. Algunos de los sistemas de gestión de paquetes más avanzados tienen la capacidad de desinstalar los paquetes recursivamente o en cascada, de forma que se eliminan todos los paquetes que dependen del paquete a desinstalar y todos los paquetes de los que el paquete a desinstalar depende, respectivamente.

Sistema operativo: Un sistema operativo es un software de sistema, es decir, un conjunto de

programas de computadora destinado a permitir una administración eficaz de sus recursos. Comienza a trabajar cuando se enciende el computador, y gestiona el hardware de la máquina desde los niveles más básicos, permitiendo también la interacción con el usuario.

Sistema base: Prototipo funcional del sistema operativo, sin interfaz gráfica y solamente con los programas mínimos necesarios para garantizar un comportamiento estable.

Suma de verificación: En la mayoría de las distribuciones GNU/Linux se utilizan técnicas para validar que los paquetes de software descargados o copiados hayan sido transferidos a su destino de forma íntegra, para ello existen algoritmos que crean una especie de código que es único e irrepetible para cada archivo. De esta forma se compara el código del sitio de origen con el código del archivo descargado para saber si la transferencia fue satisfactoria.

Sun Microsystems: Sun Microsystems es una empresa informática de Silicon Valley, fabricante de semiconductores y software. Fue constituida en 1982 por el alemán Andreas von Bechtolsheim y los norteamericanos Vinod Koshla, Bill Joy, Scott McNealy y Marcel Newman. Las siglas SUN se derivan de «Stanford University Network», proyecto que se había creado para interconectar en red las bibliotecas de la Universidad de Stanford. En ese año introducen al mercado su primera estación de trabajo que desde su inicio trabajó con el protocolo TCP/IP, protocolo sobre el cual se rige todo el tráfico de Internet.

Súper usuario: Usuario de administración en sistemas GNU/Linux, también conocido como root.

Terminal: Término derivado del inglés que significa consola.

UNIX: Unix (registrado oficialmente como UNIX®) es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969 por un grupo de empleados de los laboratorios Bell de AT&T.

Variables de entorno: Las variables de entorno son un conjunto de valores dinámicos que normalmente afectan el comportamiento de los procesos en una computadora.