UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 1

SISTEMA DE COMPILACIÓN DISTRIBUIDA DE NOVA

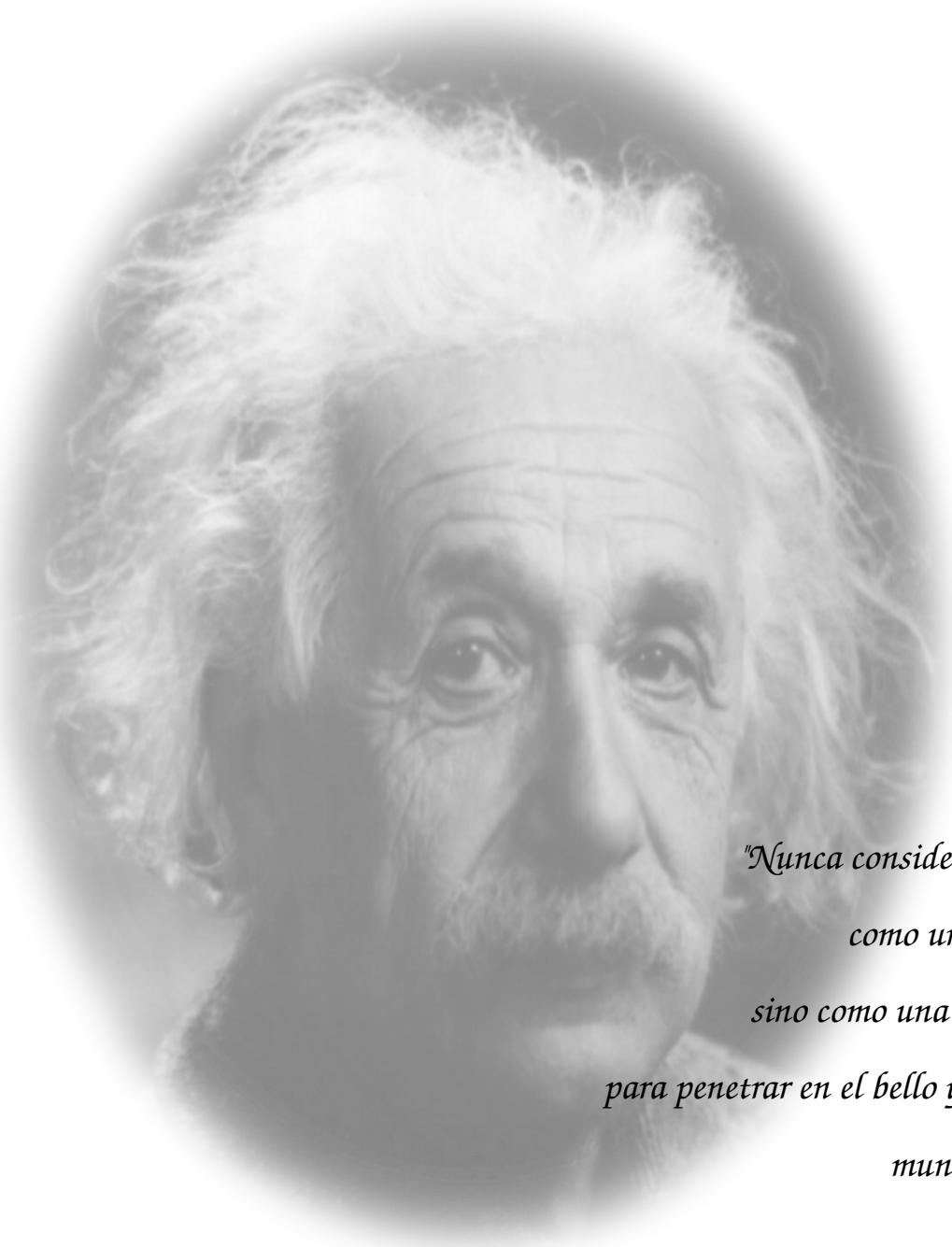
TRABAJO DE DIPLOMA EN OPCIÓN AL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: Ricardo Quevedo Mejías
Hector Pérez Baranda

Tutores: Ing. Miguel Albalat Águila
Ing. Sonia Guerrero Lambert
Ing. Jorge Luis Machín Castillo

Asesor: Lic. Dariem Pérez Herrera

La Habana, Junio 2012



"Nunca consideres el estudio

como una obligación

sino como una oportunidad

para penetrar en el bello y maravilloso

mundo del saber."

ALBERT EINSTEIN

Declaración de Autoría

Declaramos ser los únicos autores del presente trabajo y autorizamos a la Facultad 1 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

_____	_____
Ricardo Quevedo Mejías	**Hector Pérez Baranda**

_____	_____	_____
Miguel Albalat Águila	**Sonia Guerrero Lambert**	**Jorge L. Machín Castillo**

Agradecimientos

- *Agradecemos a la UCI por el aporte de conocimientos brindado durante estos 5 años de la carrera y por la oportunidad de convertirnos en ingenieros en Ciencias Informáticas.*
- *Agradecemos a la Revolución, a Fidel y a Raúl por brindarnos la opción de estudiar en una universidad de excelencia.*
- *Agradecemos a nuestros familiares por el apoyo incondicional brindado y la confianza depositada.*
- *Agradecemos a nuestros amigos por la ayuda brindada y por aguantarnos durante estos 5 años.*
- *Agradecemos a los tutores por el apoyo brindado durante todo el proceso de investigación y desarrollo de la tesis.*

Dedicatoria

A todos los que de una forma u otra nos han apoyado durante toda nuestra vida, alentándonos a seguir alcanzando nuestras metas paso a paso con la premisa de que nuestros caminos lo forjamos nosotros.

Resumen

Compilar un programa en un sistema GNU/Linux es una tarea para usuarios con experiencia en este arte, dado que cada programa tiene sus particularidades, así como cada computadora o arquitectura necesita de una compilación específica para su tipo. En Cuba existe una gran variedad de computadoras con tecnologías antiguas y que dada su necesidad aún están en servicio, por tanto, es común tener que realizar la compilación de paquetes para cada uno de estos tipos de ordenadores. En el proyecto Nova, donde se desarrolla el sistema operativo GNU/Linux Nova, que está a la vanguardia en el proceso de migración nacional, se necesita compilar ciertos códigos fuente o un repositorio completo para una arquitectura específica y no posee una herramienta que automatice este proceso realizándose manualmente.

En la actual investigación se presenta una aplicación que será capaz de lograr la compilación del repositorio de paquetes de la distribución cubana GNU/Linux Nova, de forma distribuida, para mejorar y automatizar este proceso y lograr un producto completamente genuino, contribuyendo así a la necesidad de lograr la independencia tecnológica.

Palabras claves

Compilación, Distribuida, GNU/Linux, Paquetes.

Índice de contenido

Introducción

Las computadoras digitales en un inicio ejecutaban instrucciones consistentes en códigos numéricos que señalaban a los circuitos de la máquina los estados correspondientes a cada operación, lo que se denominó lenguaje máquina, luego surgió la necesidad de escribir las instrucciones mediante claves más fáciles de recordar, que al final se traducían al lenguaje máquina.

Estas instrucciones que son escritas en un conjunto de archivos que especifican qué hacer con cada uno de ellos, para una vez ejecutados realizar una o varias tareas en una computadora, se le denomina programa.

Una aplicación necesita de varios ficheros para su funcionamiento, entre los que se encuentran los ejecutables, las bibliotecas de las que depende, las imágenes, entre otros; cuando se necesita modificar el programa o trasladarlo, la localización de cada uno de estos ficheros se puede convertir en algo muy complicado. Actualmente los sistemas de GNU/Linux se ocupan de esa complejidad, utilizando paquetes para almacenar todo lo que un programa en particular necesita para su ejecución, evitando los problemas de dependencias perdidas. Un paquete entonces es esencialmente una colección de archivos construidos en uno solo, que ofrece la posibilidad de guardar y proteger el producto durante su distribución y manipulación.

El acceso a los paquetes es a través de los Repositorios de Paquetes, que no son más que una estructura similar a una base de datos donde únicamente se guardan paquetes e información de estos. Se pueden utilizar estos repositorios para actualizar los programas instalados en nuestro sistema, o para instalar los que no se encuentren en nuestra distribución.

Los programadores cuando empaquetan un programa para los repositorios incluyen el código fuente del mismo en el paquete. El código fuente es esencialmente una lista de instrucciones para el ordenador que puede ser leída y escrita por las personas. Las computadoras pueden entender el código fuente si es

interpretado por un proceso llamado compilación, que se lleva a cabo con un programa denominado compilador, que grosso modo se encarga de la tarea de transformación del código en otro que la máquina comprende, de nombre archivo binario.

Compilar un programa en un sistema GNU/Linux es una tarea para usuarios con experiencia en este arte, dado que cada programa tiene sus particularidades, así como cada computadora o arquitectura necesita de una compilación específica para su tipo.

Actualmente en Cuba, se utiliza el sistema operativo libre GNU/Linux Nova, distribución creada con el propósito de llevar a cabo el proceso de migración nacional, dada las limitaciones establecidas por el bloqueo, las cuales impiden la obtención de *hardware*[1] y *software*[2] modernos, existe una gran variedad de computadoras con tecnologías antiguas y que dada su necesidad aún están en servicio; por este motivo es común tener que realizar la compilación de paquetes para cada uno de estos tipos de ordenadores.

En estos momentos el proceso de compilación en el proyecto Nova se hace de manera manual, lo cual hace que dicho trabajo sea tedioso, además se necesita compilar ciertos códigos fuentes o un repositorio completo para una arquitectura específica, y no posee una herramienta que automatice este proceso. Esta **situación problemática** permite plantearse el siguiente **problema científico**: ¿Cómo automatizar el proceso de compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova?

Como **objeto de estudio** para esta investigación se seleccionad el proceso de compilación de paquetes de código fuente, teniendo como **campo de acción** el proceso de compilación de paquetes de código fuente en la distribución cubana GNU/Linux Nova.

Se trazó como **objetivo general** implementar una aplicación que facilite la compilación automática y distribuida de paquetes de código fuente para Nova.

---

1 Corresponde a todas las partes tangibles de un sistema informático.
2 Corresponde a todas las partes no tangibles de un sistema informático.

El objetivo general se divide en los siguientes **objetivos específicos**:

1. Analizar el proceso de compilación de paquetes de código fuente en los sistemas GNU/Linux.
2. Desarrollar la aplicación Sistema de Compilación Distribuida de Nova.
3. Probar la aplicación Sistema de Compilación Distribuida de Nova.

Se plantea como **idea a defender** que si se desarrolla una herramienta para la compilación de código fuente de forma distribuida, es posible automatizar y disminuir la carga de trabajo del servidor durante dicho proceso en la distribución cubana GNU/Linux Nova.

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

1. Revisión bibliográfica acerca de los procesos de compilación de paquetes de código fuente, repositorios de software, aplicaciones distribuidas e interfaces de administración en servidores para la construcción de la aplicación Sistema de Compilación Distribuida de Nova.
2. Estudio de las herramientas existentes para diseñar y construir la aplicación Sistema de Compilación Distribuida de Nova.
3. Elaboración del diseño y desarrollo de la aplicación Sistema de Compilación Distribuida de Nova.
4. Elaboración, ejecución y documentación de los casos de prueba a la aplicación Sistema de Compilación Distribuida de Nova.

Los **métodos teóricos** utilizados son:

- **Analítico-sintético:** en el análisis y síntesis de la información consultada sobre las distintas herramientas de compilación de paquetes, y así elaborar ideas que se apliquen en la propuesta de solución.
- **Modelación:** en el modelado de la propuesta de solución durante la fase de análisis y diseño.

El **método empírico** utilizado es:

- **Entrevista**, como T**écnica de recopilación de información:** ésta se realizó a los que dirigen el

desarrollo de la distribución GNU/Linux Nova para lograr la recopilación de los datos necesarios para la implementación de la herramienta Sistema de Compilación Distribuida de Nova. La entre vista se enfocó en elementos como: ¿qué es compilación de paquetes?, ¿cómo se hacía la compilación de paquetes en Nova?, las herramientas de compilación que se conocían para este proceso, ¿cuáles eran las funcionalidades que deberían estar presentes en el sistema?

El presente trabajo de diploma está estructurado en 3 capítulos:

Capítulo 1: Sistema de Compilación Distribuida. Explica los principales conceptos que se tratan para lograr un mayor entendimiento del tema tratado. Se hace un estudio de las diferentes herramientas existentes para la compilación de paquetes de código fuente así como las características de un sistema distribuido para identificar funcionalidades que se le puedan agregar a la solución propuesta.

Capítulo 2: Proceso de desarrollo de la aplicación. Se definen los procesos necesarios del desarrollo de la aplicación, así como el diseño de la estructura del producto, para empezar la implementación según la metodología trazada.

Capítulo 3: Descripción y verificación de las pruebas. Se explican las tareas de implementación, los resultados obtenidos, se exponen las funcionalidades alcanzadas en el período de implementación y se diseñan, realizan y documentan las pruebas para lograr la verificación del producto.

Capítulo 1. Sistema de Compilación Distribuida

El objetivo de este capítulo consiste en plantear algunos aspectos teóricos que servirán de soporte para la elaboración del sistema. Se exponen conceptos para lograr una mejor comprensión sobre los términos tratados en el proceso de compilación de paquetes, así como un estudio del arte sobre estos sistemas de compilación en otras distribuciones de GNU/Linux. Por último, se presentan tanto la metodología como las herramientas y tecnologías a utilizar en la implementación del sistema.

1.1. Conceptos fundamentales

1.1.1. Paquete Fuente

Un paquete fuente es aquel que incluye código escrito en un lenguaje de programación, y generalmente puede ser utilizado por cualquier tipo de máquina si el código se compila de manera correcta[2.].

1.1.2. Paquete Binario

Un paquete binario es el que está construido específicamente para algún tipo de ordenador o arquitectura que puede ser *x86 (i386-i686)*, *AMD64*, *ARM*, *MIPSEL*, u otras[2.].

1.1.3. Dependencia de paquetes

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

1.1.4. Repositorio

Un repositorio es una estructura organizativa similar a las bases de datos que contiene los paquetes similares entre sí, ya sea binario o fuente, así como la información de los mismos, que están disponibles para ser descargados e instalados[2.].

1.1.5. Compilación

La compilación de un paquete es el proceso donde se genera un archivo binario que puede ser ejecutado por los ordenadores a partir de un archivo de código fuente[3.].

1.1.6. Sistema distribuido

Un sistema distribuido se define como una colección de computadores autónomos conectados por una red, y con el software distribuido adecuado para que el sistema sea visto por los usuarios como una única entidad capaz de proporcionar facilidades de computación[4.].

1.1.7. Demonio del sistema

Un demonio[3] es un tipo especial de proceso informático no interactivo, es decir, que se ejecuta de forma continua y en un segundo plano en vez de ser controlado directamente por el usuario.

1.2. Herramientas de compilación de paquetes

Existen varias herramientas que se encargan de hacer la compilación a los paquetes con sus dependencias. El estudio de algunas herramientas de compilación, hizo posible que el equipo de desarrollo del Sistema de Compilación Distribuida de Nova tenga una visión más clara de las funcionalidades que se puedan aprovechar y otras que se pueden mejorar, a la hora de la elaboración de la solución propuesta.

[3] Conocido en inglés por el término daemon, que significa **d**isk **a**nd **e**xecution **mon**itor

1.2.1. Buildd

Es una herramienta de compilación que ayuda a coordinar la reconstrucción de paquetes a través de una base de datos que mantiene una lista de paquetes, su estado, y alguna otra información de los mismos, implementadas por los desarrolladores de Debian. Incrementa la velocidad de las recompilaciones de paquetes para todas las arquitecturas que se pueden utilizar en la distro[4].

Características

El demonio del sistema de compilación automatiza la mayor parte del proceso, acción que realizaban antes los mantenedores de Debian manualmente, que era compilar las nuevas versiones de los paquetes si es que querían estar al día.

Está compuesto por una serie de scripts[5], escritos en Python y Perl, que han evolucionado, ayudando así en varias tareas a los desarrolladores, como la de velar que un paquete no se compilara dos veces, y finalmente logró que Debian se convirtiera en una distribución que se mantiene al día con los paquetes.

Las actualizaciones de los paquetes se realizan en las mismas máquinas de compilación para así asegurar la inmediata disponibilidad.

Funcionamiento

Buildd es el nombre que usualmente se le da al software, pero ésta realmente es una herramienta que está conformada por varias partes:

- **wanna build:** Herramienta central que se encarga de la gestión. Tiene una base de datos central para cada arquitectura, donde guarda la información de los paquetes.
- **buildd:** El demonio que chequea la base de datos que mantiene *wanna build* y llama a *sbuild* para que compile el paquete y lo suba para los archivos apropiados.

[4] Como se le llama coloquialmente a las distribuciones de GNU/Linux.
[5] Son programas usualmente simples, que por lo regular se almacenan en un archivo de texto plano.

- **sbuild:** Es el fichero responsable de la compilación de los paquetes en un *chroot*[6] aislado. Se asegura que todas las dependencias necesarias estén instaladas antes para la compilación y llama a las herramientas estándar de Debian para el proceso.

Actualmente los desarrolladores sólo adicionan el paquete a la base de datos para todas las arquitecturas (con el estado *Needs-Build*). Las máquinas encargadas de compilar buscarán los paquetes con este estado y lo tomarán de la lista, la cual está priorizada por los estados de prioridad como *out-of-date* o *uncompiler.* Además, para prevenir que un paquete se estanque al final de la cola, la prioridad es ajustada dinámicamente.

Si la compilación fue satisfactoria para cada arquitectura, el sistema subirá los paquetes a sus archivos correspondientes, en caso contrario el paquete pasará a un estado especial (*Build-Attempted* para las fallas que no fueron revisadas), y avisa a los responsables.

1.2.2. Open Build Service

Es una plataforma abierta y completa de desarrollo de la distribución openSUSE, que provee una infraestructura para la fácil creación y liberación de software de código abierto para la distribución y las otras distros de Linux, para diferentes arquitecturas de hardware.

Características

Presenta diversas características para diferentes destinos finales, por ejemplo, las liberaciones de los paquetes de software de openSUSE hechas en la fábrica que son transparentes para el usuario ordinario, puede obtener siempre el último paquete para su distribución, garantizando la disponibilidad a través de los diferentes *mirrors*[7] que hay alrededor del mundo.

Para otros sistemas que almacenan paquetes es de mucha ayuda ya que resuelve las dependencias de

[6] Comando en los sistemas operativos Unix que invoca un proceso o una consola interactiva con un directorio raíz especial.
[7] En computación, un espejo es una copia exacta de un conjunto de datos.

los paquetes de forma automática, si un paquete depende de otro, el primero pasará a recompilarse si su dependencia cambia. Permite el enlace con otros proyectos, por ejemplo los *parches*[8] pueden ser probados sobre su paquete desde otro proyecto. Ofrece una interfaz abierta que permite interactuar y usen sus servicios a varios clientes y servicios externos, por ejemplo *SourceForge*, entre otras.

Su característica principal para el equipo de trabajo es que es una plataforma eficiente para lograr el trabajo en equipo a través de un modelo de proyecto. No son necesarias las “granjas de compilación” de distintos hardware para la construcción de paquetes para diferentes arquitecturas y distribuciones de Linux. Presenta además gran integración para productos automáticos y creación de imagen ISO[9].

El Open Build Service está bajo la licencia GPL y puede ser instalado en cualquier máquina que tenga 2 GB de espacio en memoria. Por último, presenta una aplicación que permite correr una instancia de la plataforma o instalarla en un servidor, aunque la instalación manual también es posible pero es un poco más complicado.

Funcionamiento

Está compuesto por varios componentes oficiales y no-oficiales que hacen una plataforma completa y compacta:

Componentes Oficiales:

Build Script: Es usado por el servidor y el cliente para hacer el proceso de compilación. Puede construir en *chroot* o en un ambiente seguro como XEN o KVM.

Command Line Client: línea de comando para el cliente que es usada por *power packagers*, para la

[8] En informática, un parche consta de cambios que se aplican a un programa, para corregir errores, agregarle funcionalidad, actualizarlo, etc.

[9] Un archivo que contiene una copia idéntica de todo un sistema de archivos, generalmente grabada en un DVD, y regida por el estándar internacional ISO 9660.

solución de conflictos y compilaciones locales.

OSC sources validator: una excepción de openSUSE comander para encontrar errores comunes antes de someter un código fuente a una compilación. Esta comprobación es para cumplir con los estándares de empaquetar los paquetes en openSUSE.

OBS Server: código en el servidor que instala una instancia del Open Build Service (OBS), y puede ser instalado como una aplicación.

OBS singning daemon: es un demonio cuya función es señalar los paquetes en el servidor, es parte de la aplicación OBS.

Hermes: es un componente extra opcional para OBS. Es una solución para distribuir eventos de OBS con notificaciones a los usuarios con soporte para varias vías de notificación, que trae incluida una aplicación de configuración.

Software.o.o: es un componente extra opcional de OBS para los usuarios finales, que aún no está como un módulo para uso de propósito general.

Para la utilización del OBS se debe de tener instalado una serie de paquetes como son *y2pmbuild*, *bzip2*, *gpg*, luego se debe de configurar la herramienta *build*, con algunas opciones como dónde estará el paquete, el nombre de la base de construcción, entre otras, para que el entorno de trabajo se prepare para la construcción de los paquetes. Una vez que todo está listo para la compilación se obtiene el paquete, se arranca el build para la creación del fichero RPM[10], y una vez completada la construcción del paquete se procede a probar el mismo y seguidamente se sube para el repositorio por la herramienta y2pmbuild, esto asegura que el nuevo paquete ya esté listo para ser usado en otro proceso que dependan de él.

[10] Formato de los paquetes de algunas distros como openSUSE, Red Hat.

1.2.3. Koji

Koji es el sistema de compilación de RPM de la distribución *Fedora*, es una herramienta distribuida donde los empaquetadores utilizan el cliente para solicitar la compilación de un paquete y obtener información sobre la compilación, el servidor se encarga de la construcción de los paquetes *RPM* para arquitecturas específicas y garantiza la correcta compilación. La principal vista para interactuar con la herramienta es una aplicación web, la cual en su mayoría es de sólo lectura.

Si está logueado y presenta los permisos necesarios puede realizar las acciones: cancelar la compilación, volver a ejecutar una tarea fallida, configurar una notificación.

Si presenta privilegios administrativos podrá: editar/crear/eliminar una etiqueta, editar/crear/eliminar una tarea, activar o desactivar una compilación.

La aplicación web utiliza autenticación *SSL* y *Kerberos*, por lo se necesita en el navegador un certificado SSL en el cual confiar.

Componentes de Koji

- **Koji-Hub:** es el centro de todas las operaciones de Koji. Se trata de un servidor *XML-RPC*, se ejecuta en *mod_python* en Apache; es pasivo, ya que sólo recibe llamadas XML-RPC y se basa en los demonios de la construcción y otros componentes para iniciar la comunicación; es el único componente que tiene acceso directo a la base de datos y es uno de los dos componentes que tienen acceso de escritura al sistema de archivos.
- **Kojid:** es el demonio de construcción que se ejecuta en cada una de las máquinas de compilación. Su responsabilidad primaria es la manipulación para las solicitudes entrantes de construcción de forma adecuada. Esencialmente kojid pide a koji-hub por trabajo. La creación de imágenes de instalación es también responsabilidad de kojid, utiliza maqueta para la construcción y crea un *buildroot* nuevo para cada generación, está escrito en Python y se comunica con *koji-hub* a través

de *XML-RPC*.

- **Koji-Web:** es un grupo de scripts que envía correos en *mod_python* y utiliza el motor de plantillas *Cheetah* para proporcionar una interfaz web a Koji, actúa como un cliente de koji-hub, ofrece una interfaz visual para preformas de una cantidad limitada de la administración, expone una gran cantidad de información y también proporciona un medio para determinadas operaciones, como cancelar las generaciones.
- **Koji-client:** es una interfaz de línea de comando escrita en Python, permite al usuario consultar la mayor parte de los datos, así como realizar acciones como agregar usuarios y el inicio de construcción de las solicitudes.
- **Kojira:** es un demonio que mantiene la base de construcción actualizada, es responsable de la eliminación de la existencia de una base redundante y de limpiar después de que una solicitud de construcción se ha completado.

1.2.4. Soyuz

Soyuz es un paquete de la distribución del sistema de gestión de *Launchpad*[11] que abarca el sistema de compilación, de gestión de paquetes y publicación de los mismos. Permite a los usuarios cargar los paquetes, los que se han compilado para las distintas arquitecturas y luego publicarlo para que otros puedan descargarlos.

Cada vez que se carga un paquete para Ubuntu, o necesita crear información para ese paquete, o se descarga uno desde el archivo, se utiliza la herramienta Soyuz. Las aplicaciones también se construyen con la herramienta Soyuz.

La compilación es manejada por un número de compiladores separados físicamente en ordenadores que llevarán a cabo la compilación en un *chroot* con la arquitectura apropiada para la cual se desea construir dicho paquete.

[11] Plataforma de colaboración de software

- **Build Manager**

El nivel superior de la compilación es controlado por el demonio "*buildd manager*", el cual se responsabiliza de elegir el siguiente elemento de construcción que están "en cola" en la tabla *BuildQueue* que contiene la información de los paquetes compilados, así como los que están en proceso de compilación, se encarga también del envío de todos los archivos necesarios para los compiladores y el inicio de la construcción.

También chequea a los compiladores para obtener una cola de registro que se mostrarán. Cuando la construcción haya terminado se descargará los archivos resultantes de los compiladores para un área de prueba para luego ser procesados por *proccess-upload*.

Los compiladores de las aplicaciones son todas máquinas virtuales que están iniciadas y se reinician antes de cada construcción, ya que ejecutan código no firmado en las compilaciones. El *buildd-manager* es responsable de esta operación si determina que la construcción es virtual por *ssh-ing* y llama a una secuencia de comandos en el host de la máquina virtual.

Soyuz utiliza dos tipos de compiladores, virtuales y no virtuales. Los copiladores no virtuales actualmente sólo es utilizado por Ubuntu y el PPA[12] del equipo de Ubuntu-security. Todas las otras compilaciones que se desean se realizan en los constructores virtuales.

- **Build creation**

Cuando un fuente se acepta y se crea una compilación para él, varios prerrequisitos se deben cumplir, que la arquitectura solicitada por el fuente debe de machear con las arquitecturas soportadas por *DistroArchSeries*, que debe de haber un compilador disponible para la arquitectura solicitada y que debe de haber un fichero *chroot* disponible para los *DistroArchSeries*.

[12] **P**ersonal **P**ackage **A**rchives

Si la construcción es de una distribución (en oposición a un PPA), entonces se hace un empaquetado especial llamado *PAS*[13], el archivo es consultado para ver si se fuerza o no se le permite una construcción total para cierto paquete de cierta arquitectura. Esto es útil en situaciones, por ejemplo, donde sabemos que el paquete tiene un error en una arquitectura particular y que debe ser bloqueado hasta que el error se corrija.

- **Compiladores y chroot**

El trabajo de la administración de los compiladores, así como de los entornos chroot, en la actualidad es una función de los Administradores de Compilado para manejar las granjas de compilación. Una de las tareas importantes es la creación de imágenes chroot. Se trata de un archivo comprimido del diseño de un sistema de ficheros necesarios para iniciar una generación, cuando comienza un proceso de compilación los compiladores desempaquetan dicho archivo enviado por el administrador y el chroot dentro del mismo, cada *DistroArchSeries* tiene su propio fichero chroot, que es periódicamente actualizado por los compiladores, logrando un menor tiempo de arranque de los compiladores, ya que "*apt-get dist-upgrade*" tiene menos trabajo que hacer al inicio de cada compilación.

Hay varios scripts como *ftpmaster-tools* y *manage-chroot* que se encargan de adicionar y eliminar los comprimidos de chroot de las bibliotecas. Actualmente deben de ser ejecutados dentro del centro de datos porque no se tiene forma de lidiar con grandes volúmenes de archivos comprimidos de forma remota.

Después de que el *buildd-manager* colecte los archivos construidos una vez terminado el proceso de compilación, los pone en una cola "*incoming*", muy similar a la que *Poppy* usa para poner sus archivos de código fuente. Luego una tarea de *cron* que se ejecuta por separado utiliza los procesos de *upload* en la máquina *buildmaster* que funciona de manera casi idéntica a las que procesan los fuentes, salvo que se ejecuta con una política de carga que permite a los archivos sin firmar.

Después de la subida de los ficheros obtenidos por la compilación, se procesa el expediente de

[13] Paquete Arco Específico.

construcción que sirve como un vínculo entre la fuente y su binario, esto se hace a través de consultas SQL usando Soyuz.

Valoración del estudio de las herramientas de compilación.

De las cuatro herramientas de compilación estudiadas (Buildd, Open Build Service, Koji y Soyuz) ninguna cumple con las características del país y del proyecto porque tienen como limitantes que algunas requieren de la conexión a Internet para subir los fuentes a compilar como Open Build Service, Koji y Soyuz, en Buildd la versión liberada es obsoleta y descontinuada, y como la mayoría de estas herramientas realizan la compilación en Internet, el proyecto Nova desconoce el estado real de compilación de cada paquete, lo que impide lograr la soberanía tecnológica.

1.3. Métodos de comunicación

Una de las características vistas en los sistemas de compilación analizados, fue la utilización del patrón arquitectónico cliente servidor y que algunos son distribuidos, y es precisamente el ser distribuido lo que agiliza el trabajo en gran medida, ya que ayuda a liberar una gran carga de trabajo del servidor, así como acortar el tiempo total de compilación de todos los paquetes al haber varias instancias realizando ese proceso.

Para la conformación de una estructura como la del cliente-servidor es necesario que se establezca un mecanismo de comunicación que permita una interacción constante de información entre ambas partes. En el caso de la solución propuesta se utilizarán como nodos de compilación los ordenadores que se encuentren en la subred del servidor, el cual es la computadora donde se gestiona la disposición de los paquetes en el repositorio.

A continuación se hace un estudio sobre los métodos de comunicación más utilizados para este tipo de aplicación.

1.3.1. XMPP

XMPP[14] anteriormente llamado jabber. Es un protocolo abierto y extensible basado en *XML*[15], originalmente ideado para mensajería instantánea. Con este protocolo queda establecida una vía de intercambio de datos XML, que pueden ser usadas en aplicaciones de mensajería instantánea, heredando la adaptabilidad y sencillez de uso del XML.

La red XMPP está basada en servidores, pero descentralizada por diseño; no hay ningún servidor central, como sucede con otros servicios.

Ventajas

- **Descentralización:** Cualquiera puede poner en marcha su propio servidor XMPP, sin que haya ningún servidor central.
- **Estándares abiertos:** El protocolo XMPP se ha formalizado como una tecnología de mensajería instantánea estándar, el desarrollo de esta tecnología no está ligado a ninguna empresa en concreto y no requiere el pago de regalías[1.].
- **Historia:** Las tecnologías XMPP llevan usándose desde 1998. Existen múltiples implementaciones de los estándares XMPP para clientes, servidores, componentes y bibliotecas.
- **Seguridad:** Los servidores XMPP pueden estar aislados de la red pública XMPP, y poseen robustos sistemas de seguridad (como SASL y TLS)[5.].
- **Flexibilidad:** Se pueden hacer funcionalidades a medida sobre XMPP; para mantener la interoperabilidad, las extensiones más comunes son gestionadas por la XMPP Software Foundation.

[14] Extensible Messaging and Presence Protocol, Protocolo extensible de mensajería y comunicación de presencia
[15] e**X**tensible **M**arkup **L**anguage (Lenguaje de marcas extensible).

Desventajas

- **Sobrecarga de datos de presencia:** Cerca de un 70% del tráfico entre servidores son datos de presencia [6.], y cerca de un 60% de estos son transmisiones redundantes[7.].
- **Escalabilidad:** XMPP también sufre el mismo problema de redundancia en los servicios de *chatroom* y de suscripción.
- **Sin datos binarios:** XMPP es codificado como un único y largo documento XML, lo que hace imposible entregar datos binarios sin modificar. De todas formas, las transferencias de archivos se han solucionado usando otros protocolos como HTTP[16].

1.3.2. SSH

SSH[17] es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite el control de una máquina a través de un intérprete de comando. Usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible y ninguna tercera persona pueda descubrir el usuario y contraseña de la conexión ni lo que se escribe durante toda la sesión[8.].

Ventajas

- Es usado en el acceso de escritura autenticado.
- Es sencillo de habilitar.
- El acceso es seguro estando todas las transferencias cifradas y autenticadas.
- Comprime los datos lo más posible antes de transferirlos.

[16] Hypertext Transfer Protocol (*protocolo de transferencia de* hipertexto)
[17] **S**ecure **Sh**ell, intérprete de orden segura

Desventajas

- No da la posibilidad de acceso anónimo.
- Todos han de tener configurado un acceso SSH al servidor, incluso aunque sea con permisos de solo lectura, lo que no lo hace recomendable para soportar proyectos abiertos.

1.3.3. Socket

Un socket, es un método para la comunicación entre un programa del cliente y un programa del servidor en una red. Se define como el punto final en una conexión. Estos se crean y se utilizan con un sistema de peticiones o de llamadas de función a veces llamados interfaz de programación de aplicación de sockets (API, *Application Programming Interface*)[9.].

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente". El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor"[10.].

Ventajas

- El protocolo que más utiliza para su implementación es el TCP/IP.
- Son orientados a la conexión.
- Se garantiza la transmisión de todos los octetos sin errores ni omisiones.
- Se garantiza que todo octeto llegará a su destino en el mismo orden en que se ha transmitido.

Desventajas

- Requiere la definición de un protocolo propio, ya que solo sabe enviar y recibir arreglos de bytes, luego es necesario diseñar y programar la lógica que le impregna semántica a esos bytes y los convierte en un protocolo nuevo.
- Posee mayor dificultad al trabajar ya que la libertad de acción se convierte en un arma de doble

filo, por una parte abre un mundo de posibilidades y flexibilidad, pero por el otro lado no se responsabiliza de los detalles del mecanismo, incluyendo el manejo completo de errores.

1.3.4. FTP

Una vez que se haya compilado el paquete, se necesita que éste sea colocado en una dirección, la cual será la ubicación del repositorio de donde luego se pueda acceder a él. Para esta tarea se decidió investigar acerca del uso del protocolo *FTP*[18], protocolo de red para la transferencia de archivos entre sistemas conectados a una red, basado en la arquitectura cliente-servidor.

El servicio FTP es ofrecido por la capa de aplicación del modelo de capas de red TCP/IP al usuario, utilizando normalmente el puerto de red 20 y 21, y pensado principalmente para ofrecer la máxima velocidad en la conexión.

La conexión de datos es bidireccional, es decir, se puede usar simultáneamente para enviar y para recibir, y no tiene por qué existir todo el tiempo que dura la conexión FTP.

Acceso anónimo

Los servidores FTP anónimos ofrecen sus servicios libremente a todos los usuarios, permiten acceder a sus archivos sin necesidad de tener un 'USER ID' o una cuenta de usuario. Es la manera más cómoda fuera del servicio web de permitir que todo el mundo tenga acceso a cierta información sin que para ello el administrador de un sistema tenga que crear una cuenta para cada usuario.

Si un servidor posee servicio 'FTP *anonymous*' solamente con teclear la palabra «anonymous», cuando pregunte por el usuario tendrá acceso a ese sistema. No se necesita ninguna contraseña preestablecida, aunque deberá introducir una sólo para ese momento, normalmente se suele utilizar la dirección de correo electrónico propia.

[18] (**F**ile **T**ransfer **P**rotocol) Protocolo de Transferencia de Archivos.

Solamente con eso se consigue acceso a los archivos del FTP, aunque con menos privilegios que un usuario normal. Normalmente solo se podrá leer y copiar los archivos que sean públicos, así indicados por el administrador del servidor al que se quiera conectar.

Normalmente, se utiliza un servidor FTP anónimo para depositar grandes archivos que no tienen utilidad si no son transferidos a la máquina del usuario, como por ejemplo programas, y se reservan los servidores de páginas web (HTTP) para almacenar información textual destinada a la lectura en línea.

Acceso de usuario

Si se desea tener privilegios de acceso a cualquier parte del sistema de archivos del servidor FTP, de modificación de archivos existentes, y de posibilidad de subir archivos, generalmente se suele realizar mediante una cuenta de usuario. En el servidor se guarda la información de las distintas cuentas de usuario que pueden acceder a él, de manera que para iniciar una sesión FTP debemos introducir una autenticación o login y una contraseña o *password* que autentica al usuario unívocamente.

Acceso de invitado

El acceso sin restricciones al servidor que proporcionan las cuentas de usuario implica problemas de seguridad, lo que ha dado lugar a un tercer tipo de acceso FTP denominado invitado (*guest*), que se puede contemplar como una mezcla de los dos anteriores.

La idea de este mecanismo es la siguiente: se trata de permitir que cada usuario se conecte a la máquina mediante su login y su password, pero evitando que tenga acceso a partes del sistema de archivos que no necesita para realizar su trabajo, de esta forma accede a un entorno restringido, algo muy similar a lo que sucede en los accesos anónimos, pero con más privilegios.

Modos de conexión del cliente FTP

FTP admite dos modos de conexión del cliente. Estos modos se denominan activo (o Estándar, o PORT,

debido a que el cliente envía comandos tipo PORT al servidor por el canal de control al establecer la conexión) y pasivo (o PASV, porque en este caso envía comandos tipo PASV). Tanto en el modo Activo como en el modo Pasivo, el cliente establece una conexión con el servidor mediante el puerto 21, que establece el canal de control.

1. **Modo activo**

En modo Activo, el servidor siempre crea el canal de datos en su puerto 20, mientras que en el lado del cliente el canal de datos se asocia a un puerto aleatorio mayor que el 1024. Para ello, el cliente manda un comando PORT al servidor por el canal de control indicándole ese número de puerto, de manera que el servidor pueda abrirle una conexión de datos por donde se transfieren los archivos y los listados, en el puerto especificado.

Lo anterior tiene un grave problema de seguridad, y es que la máquina cliente debe estar dispuesta a aceptar cualquier conexión de entrada en un puerto superior al 1024, con los problemas que ello implica si tenemos el equipo conectado a una red insegura como Internet. De hecho, los cortafuegos que se instalen en el equipo para evitar ataques comúnmente rechazan esas conexiones aleatorias. Para solucionar esto se desarrolló el modo pasivo.

2. **Modo pasivo**

Cuando el cliente envía un comando PASV sobre el canal de control, el servidor FTP le indica por el canal de control, el puerto (mayor a 1023 del servidor. Ej.: 2040) al que debe conectarse el cliente. El cliente inicia una conexión desde el puerto siguiente al puerto de control (ej.: 1036) hacia el puerto del servidor especificado anteriormente (ej.: 2040).

Antes de cada nueva transferencia tanto en el modo Activo como en el Pasivo, el cliente debe enviar otra vez un comando de control (PORT o PASV, según el modo en el que haya conectado), y el servidor recibe esa conexión de datos en un nuevo puerto aleatorio (si está en modo pasivo) o por el puerto 20 (si está en modo activo). En el protocolo FTP existen 5 tipos de transferencia en ASCII y en binarios.

Tipos de transferencia de archivos en FTP

Es importante conocer cómo debemos transportar un archivo a lo largo de la red. Si no se utilizan las opciones adecuadas es posible destruir la información del archivo. Por eso, al ejecutar la aplicación FTP, se acuerda utilizar uno de estos comandos (o poner la correspondiente opción en un programa con interfaz gráfica):

1. **Tipo ascii:** adecuado para transferir archivos que sólo contengan caracteres imprimibles (archivos ASCII, no archivos resultantes de un procesador de texto), por ejemplo páginas HTML, pero no las imágenes que puedan contener.
2. **Tipo binario:** este tipo es usado cuando se trata de archivos comprimidos, ejecutables para PC, imágenes, archivos de audio.

Valoración del estudio de los métodos de comunicación.

En el caso de la solución propuesta se utilizará el tipo de transferencia binaria y el modo de conexión pasiva con acceso de invitado, así como el método socket para la comunicación entre los distintos nodos del sistema, el cual permite una mayor abstracción en cuanto a comunicación entre programas se refiere, es capaz de permitir el intercambio de información en ambas direcciones de manera segura sin necesidad de la utilización de claves o permiso de algún tipo, como era el caso con la utilización de ssh, permite una mayor flexibilidad y sencillez de comunicación sin la necesidad de utilizar protocolos más complejos y que no están diseñados para la utilización del patrón arquitectónico cliente-servidor, como por ejemplo el XMPP; y como seguridad en el envío de datos a través de sockets se utilizará SSL[19], el cual permite la transferencia de datos cifrados y solamente se puede acceder a la información transmitidas por las partes que poseen el certificado de autorizo.

[19] Socket Security Layer

1.4. Gestor de Bases de Datos

Los sistemas de gestión de bases de datos (SGBD)[20] son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Su propósito general es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para una organización. A continuación se analizan 3 de los principales gestores de bases de datos libres existentes para escoger cuál es el más indicado según sus necesidades para la herramienta Sistema de Compilación Distribuida de Nova. Los gestores a analizar son PostgreSQL, Mysql y Sqlite.

1.4.1. PostgreSQL

PostgreSQL es un sistema de gestión de base de datos relacional y libre, publicado bajo la licencia BSD[21]. Como muchos proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabaja de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. Dicha comunidad es denominada el PGDG[22].

Características

- Implementación del estándar SQL92/SQL99.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.
- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor

[20] En inglés database management system, abreviado DBMS
[21] BSD (Berkeley Software Distribution) permite el uso de código fuente en software no libre.
[22] PostgreSQL Global Development Group

de bases de datos se le incluye entre los gestores objeto-relacionales.

- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.
- Mediante un sistema denominado MVCC[23] PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último a lo que se le hizo "commit".

Ventajas

- **Instalación ilimitada**

  Es frecuente que las bases de datos comerciales sean instaladas en más servidores de lo que permite la licencia. Algunos proveedores comerciales consideran a esto la principal fuente de incumplimiento de licencia. Con PostgreSQL, nadie puede demandarlo por violar acuerdos de licencia, puesto que no hay costo asociado a la licencia del software.

  Esto tiene varias ventajas adicionales: modelos de negocios más rentables con instalaciones a gran escala, no existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento, flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.

- **Mejor soporte que los proveedores comerciales**

  Además de las ofertas de soporte, se tiene una importante comunidad de profesionales y entusiastas de PostgreSQL de los que su compañía puede obtener beneficios y contribuir.

- **Ahorros considerables en costos de operación**

  Nuestro software ha sido diseñado y creado para tener un mantenimiento y ajuste mucho menor que los productos de los proveedores comerciales, conservando todas las características, estabilidad y rendimiento. Además de esto, los programas de entrenamiento son reconocidamente mucho más costo-efectivos, manejables y prácticos en el mundo real que aquellos de los principales proveedores comerciales.

[23] Acceso concurrente multiversión, por sus siglas en inglés

- **Estabilidad y confiabilidad legendarias**
  En contraste a muchos sistemas de bases de datos comerciales, es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas en varios años de operación de alta actividad. Ni una sola vez. Simplemente funciona.
- **Multiplataforma**
  PostgreSQL está disponible en casi cualquier Unix (34 plataformas en la última versión estable), y una versión nativa de Windows está actualmente en estado beta de pruebas.
- **Diseñado para ambientes de alto volumen**
  PostgreSQL usa una estrategia de almacenamiento de filas llamada MVCC para conseguir una mucho mejor respuesta en ambientes de grandes volúmenes. Los principales proveedores de sistemas de bases de datos comerciales usan también esta tecnología, por las mismas razones.
- **Herramientas gráficas de diseño y administración de bases de datos**
  Existen varias herramientas gráficas de alta calidad para administrar las bases de datos (pgAdmin, pgAccess) y para hacer diseño de bases de datos (Tora, Data Architect).

Desventajas

- Es más lento en inserciones y actualizaciones, ya que cuenta con cabeceras de intersección.
- No posee amplio soporte en línea.
- La sintaxis de algunos de sus comandos o sentencias no es nada intuitiva.
- Es vulnerable sin una adecuada configuración y protección.

1.4.2. MySQL

MySQL es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso. Está desarrollado en su mayor parte en ANSI

C[24]. MySQL es patrocinado por una empresa privada, que posee el copyright de la mayor parte del código.

Características

1. Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multihilo.
2. Soporta gran cantidad de tipos de datos para las columnas.
3. Dispone de API's en gran cantidad de lenguajes (C, C++, Java, PHP, etc).
4. Gran portabilidad entre sistemas.
5. Gestión de usuarios y password, manteniendo un muy buen nivel de seguridad en los datos.
6. Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
7. Disponibilidad en gran cantidad de plataformas y sistemas.
8. Posibilidad de selección de mecanismos de almacenamiento que ofrecen diferente velocidad de operación, soporte físico, capacidad, distribución geográfica, transacciones.
9. Transacciones y claves foráneas.
10. Conectividad segura.
11. Replicación.
12. Búsqueda e indexación de campos de texto.

Características adicionales

- Usa GNU Automake, Autoconf, y Libtool para portabilidad.
- Usa tablas en disco b-tree para búsquedas rápidas con compresión de índice.
- Tablas hash en memoria temporales.
- El código MySQL se prueba con Purify[25] así como con Valgrind[26].
- Completo soporte para operadores y funciones en cláusulas select y where.
- Completo soporte para cláusulas group by y order by, soporte de funciones de agrupación.

[24] es un estándar publicado por el Instituto Nacional Estadounidense de Estándares (ANSI), para el lenguaje de programación C.
[25] un detector de memoria perdida comercial
[26] una herramienta GPL

- Ofrece un sistema de contraseñas y privilegios seguro mediante verificación basada en el host y el tráfico de contraseñas está cifrado al conectarse a un servidor.
- MySQL Enterprise soporta gran cantidad de datos. MySQL Server tiene bases de datos de hasta 50 millones de registros.
- Se permiten hasta 64 índices por tabla (32 antes de MySQL 4.1.2). Cada índice puede consistir desde 1 hasta 16 columnas o partes de columnas. El máximo ancho de límite son 1000 bytes (500 antes de MySQL 4.1.2).
- Los clientes se conectan al servidor MySQL usando sockets TCP/IP en cualquier plataforma.
- MySQL contiene su propio paquete de pruebas de rendimiento proporcionado con el código fuente de la distribución de MySQL.

Ventajas

- Posee una gran velocidad al realizar las operaciones, lo que hace que tenga un buen rendimiento.
- Posee bajo costo en requerimientos para la elaboración de bases de datos, ya que debido a su bajo consumo puede ser ejecutado en una máquina con escasos recursos.
- Posee una gran facilidad de configuración e instalación.
- Soporta gran variedad de sistemas operativos.
- Posee baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en que está.
- Soporte completo para cláusulas, funciones, tipos de datos y comandos estándar y extendidos del estándar SQL.

Desventajas

- Un gran porcentaje de las utilidades no están documentadas.
- No es intuitivo.
- Actualmente, el soporte para disparadores es básico, por lo tanto hay ciertas limitaciones en lo que

puede hacerse con ellos.

- Cuando MySQL maneja la Integridad referencial, con tablas no transaccionales de tipo MyISAM, aunque admite la declaración de claves ajenas o foráneas en la creación de tablas, internamente no las trata de forma diferente al resto de campos.
- Los privilegios para una tabla no se eliminan automáticamente cuando se borra una tabla. Debe usarse explícitamente un comando REVOKE para quitar los privilegios de una tabla.

1.4.3. SQLite

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID[27], es un proyecto de dominio público, el cual implementa una pequeña biblioteca de aproximadamente 500kb, programado en el lenguaje C, de dominio público, totalmente libre.

A diferencia de los sistema de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB(un texto con un máximo de 65535 caracteres, sirve para almacenar datos binarios de cualquier tipo y de longitud variable, como por ejemplo un archivo ejecutable, archivos de imágenes, cadenas de bytes, etc.).

[27] (ACID**)** **A**tomicity, **C**onsistency, **I**solation and **D**urability, conjunto de características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

La base de datos se almacena en un único fichero a diferencia de otros DBMS que hacen uso de varios archivos. SQLite emplea registros de tamaño variable de forma tal que se utiliza el espacio en disco que es realmente necesario en cada momento. El código fuente está pensado para que sea entendido y accesible por programadores promedio.

Ventajas:

- **Tamaño**: SQLite tiene una pequeña memoria y una única biblioteca es necesaria para acceder a bases de datos, lo que lo hace ideal para aplicaciones de bases de datos incorporadas.
- **Rendimiento de base de datos**: SQLite realiza operaciones de manera eficiente y es más rápido que MySQL y PostgreSQL.
- **Concurrencia:** Varios procesos o hilos pueden acceder a la misma base de datos sin problemas. Varios accesos de lectura pueden ser servidos en paralelo. Un acceso de escritura sólo puede ser servido si no se está sirviendo ningún otro acceso concurrentemente.
- **Portabilidad**: se ejecuta en muchas plataformas y sus bases de datos pueden ser fácilmente portadas sin ninguna configuración o administración.
- **Estabilidad**: SQLite es compatible con ACID.
- **SQL**: implementa un gran subconjunto de la ANSI – 92 SQL estándar, incluyendo subconsultas, generación de usuarios, vistas y triggers.
- **Interfaces**: cuenta con diferentes interfaces del API, las cuales permiten trabajar con C++, PHP, Perl, Python, Ruby, Tcl, groovy, etc.
- **Costo**: SQLite es de dominio público, y por tanto, es libre de utilizar para cualquier propósito sin costo y se puede redistribuir libremente.
- **Configuración:** De la forma en que fue creado y diseñado SQLite, no necesita ser instalado, ni prendido, reiniciado o apagado un servidor, e incluso configurarlo. Esta cualidad permite que no haya un administrador de base de datos para crear las tablas, vistas, asignar permisos, o bien la adopción de medidas de recuperación de servidor por cada caída del sistema.

Limitaciones

- **Limitaciones en Where**: esta limitación está dada por el soporte para clausuras anidadas.
- **Falta de Clave Foránea**: permite la definición de la cláusula de clave foránea pero en la práctica no hace chequeo de la misma.
- **Limitaciones en la implementación**: no permite el trabajo con funciones ni cursores ni procedimientos.
- **Falta de documentación en español**: si bien ya se cuenta con una comunidad latino americana de SQLite, sería importante encontrar mucha más documentación, libros, review, etc. como muchos otros motores de bases de datos cuentan hoy en día.
- No soporta alta concurrencia.
- No soporta la estructura cliente-servidor.

Valoración del estudio de los gestores de bases de datos.

Se utiliza PostgreSQL como gestor de base de datos debido a que en comparación con la versión liberada de MySQL es mejor en la gestión de grandes volúmenes de datos. Además, SQLite no permite la definición de funciones, cursores y procedimientos, elementos fundamentales en la programación de la base de datos; esto sí se realiza en PostgreSQL.

1.5. Metodología de desarrollo

El desarrollo de software es una actividad que se debe de controlar, para obtener mayores resultados, así como lograr hacer un trabajo organizado, el cual cumpla con los requisitos planteados. Para esto ha surgido una solución, las metodologías de desarrollo.

Una metodología de desarrollo de software no es más que un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo en sistemas de información. [11.] Es un conjunto de procedimientos, técnicas, herramientas que ayuda a los desarrolladores a realizar software.

Las metodologías indican las tareas que son las actividades fundamentales en las que se dividen los procesos, también define la forma en que se ejecutan estas tareas, para que al final salga el producto deseado.

Existen dos tipos de metodología:

Pesadas o tradicionales: Hace énfasis en la planificación del proceso de desarrollo, imponiendo una mayor disciplina, luego de que todo esté en orden se comienza el desarrollo. Se emplea en el desarrollo de sistemas grandes, lo cual hace que no se adapte a entornos de trabajos cambiantes, donde los requisitos o no se pueden predecir o están en constante cambio.

Ágiles: Apuestan por un desarrollo de software más rápido, ya que se adaptan con mayor facilidad a cualquier cambio que se pueda presentar, intenta evitar los caminos de las metodologías pesadas, enfocándose más en la producción rápida de software.

Debido que el equipo de desarrollo está compuesto por 2 personas, y el tiempo de desarrollo es relativamente corto, así como que los requisitos son cambiantes, se decide el uso de una metodología ágil. Dado que el desarrollo de la herramienta está enmarcado en el proyecto Nova, se utilizará la metodología que se utiliza en el mismo, ***OpenUp***, la cual es una metodología ágil muy idónea para las entregas de productos en pequeños espacios de tiempo y ayuda a la creación, distribución y mantenimiento de las aplicaciones de software.

Características de la metodología OpenUp

OpenUp/Basic es un *framework* de procesos de desarrollo de programas de código abierto, que con el tiempo espera cubrir un amplio conjunto de necesidades en el campo del desarrollo de programas. Permite un abordaje ágil al programa en análisis con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles y tareas. Es un proceso interactivo de desarrollo de software simplificado, completo y extensible; para pequeños equipos de

desarrollo, que valoran los beneficios de la colaboración y los involucrados, con el resultado del proyecto, por encima de formalidades innecesarias.

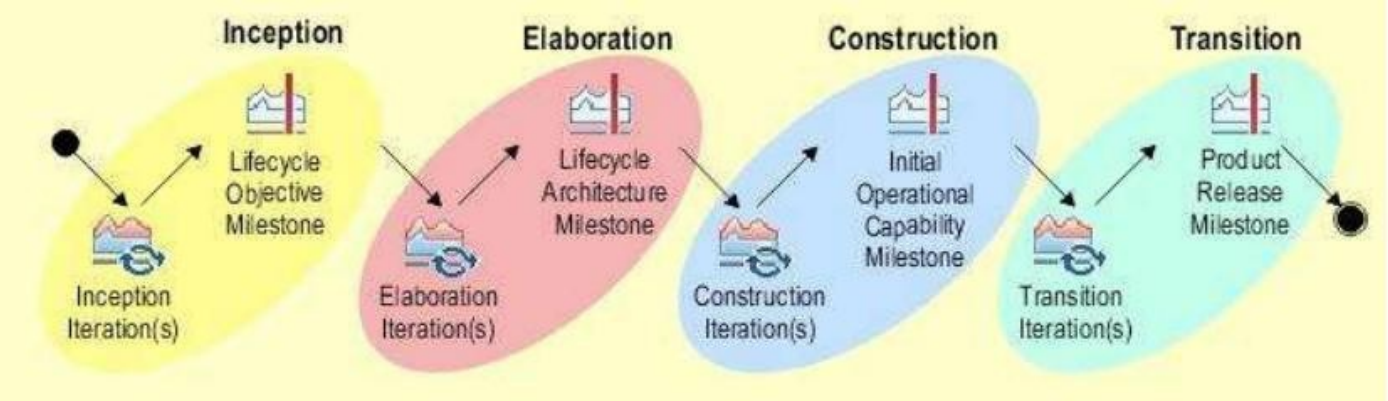


Ilustración 1.1: Ciclo de Vida de la metodología OpenUp

En la Ilustración 1.1 se puede observar las cuatro fases de desarrollo que propone la metodología, las cuales se definen a continuación:

1. **Fase de Concepción:** realizar los planteamientos del problema; elaborar la justificación; definir objetivo general y objetivos específicos; obtener toda la información relacionada con el proyecto y capturar las necesidades de los clientes en los objetivos del ciclo de vida para el proyecto.
2. **Fase de Elaboración:** definir los riesgos significativos para la arquitectura y establecer la base para la elaboración de la arquitectura del sistema.
3. **Fase de Construcción:** diseñar, implementar y realizar las pruebas de las funcionalidades para realizar un sistema completo y completar el desarrollo del sistema basado en la arquitectura definida.
4. **Fase de Transición:** asegurar que el sistema sea entregado a los usuarios y evaluar la funcionalidad y escena del último entregable de la fase de construcción.

1.6. Herramientas

La herramienta que se selecciona para el desarrollo de la aplicación Sistema de Compilación Distribuida de Nova es el IDE[28] libre de programación Eclipse, está bajo la licencia GPL, es una herramienta de código abierto multiplataforma, dispone de un editor de texto con resaltado de sintaxis, la compilación es en tiempo real, a través de plugins[29] libremente disponibles es posible añadir control de versiones con Subversion, posibilitando el trabajo colaborativo. Como plugin de Python para Eclipse se emplea PyDev con ventajas como completado de código, resaltado de sintaxis, análisis de código, un depurador y una consola interactiva.

Como herramienta CASE[30] se utilizó el Visual Paradigm, que es una herramienta UML[31] profesional que soporta el ciclo de vida completo de desarrollo del software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El lenguaje de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad y generación automática de código. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

Visual Paradigm soporta los principales estándares de la industria tales como SysML, BPMN, XMI, entre otros. Ofrece un completo conjunto de herramientas de los equipos de desarrollo de software necesario para los requisitos de la captura, software de planificación, la planificación de controles, el modelado de clases y datos, entre otros[12.].

1.7. Lenguajes

Uno de los lenguajes que se utilizan es Python, que es de alto nivel, cuya filosofía hace hincapié en una sintaxis muy limpia y que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma ya que soporta orientación a objetos, programación imperativa y programación funcional.

[28] Entorno de Desarrollo Integrados
[29] Aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy especifica
[30] Computer Aided Software Engineering / Ingeniería de Software Asistida por Computadora
[31] UML: Lenguaje Unificado de Modelado (por sus siglas en inglés, Unified Modeling Language)

Es un lenguaje interpretado, detecta el tipo de las variables en tiempo de ejecución y es multiplataforma.

Otro lenguaje fue Perl el cual es originalmente desarrollado para la manipulación de texto y que ahora es utilizado para un amplio rango de tareas incluyendo administración de sistemas, desarrollo web, programación en red, desarrollo de GUI y más. Sus principales características son que es fácil de usar, soporta tanto la programación estructurada como la programación orientada a objetos y la programación funcional, tiene incorporado un poderoso sistema de procesamiento de texto y una enorme colección de módulos disponibles. Es un lenguaje imperativo, con variables, expresiones, asignaciones, bloques de código delimitados por llaves, estructuras de control y subrutinas. Además tiene muchas funciones integradas para tareas comunes y para acceder a los recursos del sistema. Se emplea este lenguaje debido a que Perl es el utilizado en la plataforma en la que se integró el Sistema de Compilación Distribuida de Nova.

1.8. Conclusiones del Capítulo.

En el concluido capítulo se abordaron conceptos relacionados con la investigación como por ejemplo repositorios, paquetes binarios y fuentes; y se realizó un análisis de algunos de los sistemas de compilación de paquetes más usados o los que tenían implementación muy parecidas a las deseadas, lo cual sirvió como base para proponer una solución acorde con las necesidades actuales del sistema operativo GNU/Linux Nova. Se realizó un estudio de los principales gestores de base de datos y libres y atendiendo a las necesidades de la herramienta Sistema de Compilación Distribuida de Nova se decidió usar PostgreSQL. Además se realizó la selección de la metodología que guiará el proceso de desarrollo, así como las herramientas y lenguajes a utilizar en el mismo. Para lograr la arquitectura cliente-servidor de una aplicación distribuida es más viable hacer uso del método de comunicación a través de sockets.

Capítulo 2. Proceso de Desarrollo de la Aplicación

El siguiente capítulo se enfoca en plantear la vía a seguir para la realización de la solución propuesta que resuelve el problema planteado, siguiendo la metodología OpenUp para llegar a tener un producto con buena calidad teniendo en cuenta las prácticas y los principios que contribuyan a agilizar el proceso y cumpla con las características de un producto en el nivel 2 del CMMI[32].

Los objetivos trazados son:

1. Explicar la solución propuesta.
2. Definir los flujos de trabajo que permitirán conocer las actividades a realizar, los roles a cumplir por cada uno de los miembros del equipo, los artefactos que se obtendrían de la implementación de la solución y los requerimientos funcionales y no funcionales.

2.1. Modelado del sistema

A continuación se describe en detalles la construcción de la aplicación Sistema de Compilación Distribuida de Nova, se realiza la propuesta del sistema y se analizan los requerimientos funcionales y no funcionales.

2.1.1. Solución Propuesta

Como se ha abordado hasta este punto de la investigación, para darle solución al problema tratado se propone el desarrollo de la aplicación Sistema de Compilación Distribuida de Nova que logre la compilación automática y distribuida de paquetes de código fuente para la distribución cubana GNU/Linux Nova.

La aplicación formará parte de los módulos instalables de la personalización de Nova para administración de servidores, llamada Nova Server, el cual utiliza Zentyal, (plataforma anteriormente conocida como ebox

[32] Capability Maturity Model Integration

Platform). Puede actuar gestionando la infraestructura de red, como puerta de enlace a Internet gestionando las amenazas de seguridad, como servidor de oficina, como servidor de comunicaciones unificadas o una combinación de estas. Es una aplicación web que usa el servidor web Apache y escrito en Perl orientado a objetos con mejoras visuales en javascript.

La aplicación se desarrolla con una arquitectura cliente-servidor en la cual la estación principal se conecta a la herramienta Hergire[33] en busca de los repositorios disponibles para compilar, dándole la oportunidad al administrador de insertar manualmente uno o varios que desee compilar específicamente; la información de los paquetes de estos repositorios es insertada en la base de datos donde se generan los paquetes que están listos para compilarse, mostrándose los mismos al usuario brindándole la oportunidad de seleccionar aquellos que desee enviar a compilar; estos paquetes seleccionados son enviados a compilar a los nodos de compilación, en dichos nodos se descarga el código fuente del paquete enviado, se compila utilizando *pbuilder,* donde primeramente se genera una base de compilación con los paquetes de un repositorio externo que será el entorno de compilación y una vez terminado la construcción del paquete este es almacenado en un servidor FTP designado, y el estado del paquete, así como los errores que ocurran durante la compilación o la subida del paquete al FTP son enviados a la estación principal almacenándose esta información en la base de datos.

El administrador del sistema registrará la dirección de los repositorios de código fuente que se utilizarán para la extracción de los paquetes a compilar, así como los repositorios binarios que se utilizarán como base para la construcción del mismo y para qué arquitectura se compilará. Se empleará como apoyo una base de datos en la cual se guardará un registro del estado de los paquetes antes, durante y después del proceso de compilación. Los estados de los paquetes que se manejarán en el sistema son:

- **Paquetes listos para compilar (ready):** son aquellos paquetes que poseen todas sus dependencias en estado compilado (done), externo (outside) o enfermo (sick).
- **Paquete compilándose (building):** es el paquete que se está compilando.

[33] **Her**ramienta de **G**estión **I**ntergral de **Re**positorios

- **Paquete compilado o hecho (done):** es el paquete que se compiló y tiene todas sus dependencias en estado *enfermo* o *hecho*.
- **Paquete fallido (failed):** es el paquete que falló durante su compilación o la subida del compilado al FTP, pero este paquete no se almacena en la base de datos como fallido sino como externo con bandera en 1 o estado externo con bandera en 2, el cual no se vuelve a enviar a compilar hasta que el mismo haya sido actualizado, o sys dependencias. El otro caso es el fallido con bandera en 5 al inicio de la carga de la base de datos, es cuando en una primera carga del repositorio a compilar el paquete es almacenado y en una segunda carga del mismo repositorio no aparece, si hay paquetes que dependen de él no puede ser eliminado y adquiere este estado.
- **Paquete enfermo (sick):** es el paquete que se compiló y tiene al menos una dependencia en estado externo, entonces, cuando sus dependencias están en estado *hecho* o *enfermo* vuelve al estado listo para compilar.
- **Paquete externo (outside):** es el paquete que no se encuentra en el repositorio a compilar, por lo que hay que buscarlo en un repositorio externo. Además de esto, existen 4 clasificaciones:

1. **Estado externo con bandera en 1:** es cuando el paquete se compila por primera vez y la compilación falla.
2. **Estado externo con bandera en 2:** es cuando el paquete se compila por segunda vez y vuelve a fallar la compilación.
3. **Estado externo con bandera en 3:** es cuando el paquete depende de un paquete fuente fallido con bandera en 5.
4. **Estado externo con bandera en 4:** es cuando el paquete depende de otro paquete fuente que no existe.

La solución propuesta da la posibilidad de generar un reporte del proceso de compilación, donde se informará el total de paquetes enviados a compilar, cuántos de ellos se compilaron, cuántos están compilándose, cuántos de ellos fallaron y cuáles son los errores que existieron, cuántos están en externos y enfermos por compilación. Además, brinda la posibilidad de que si el sistema falla por alguna razón

desconocida recupere los datos que tenía durante el proceso de construcción del repositorio.

2.1.2. Actor del Sistema

Actor	Objetivo
Administrador del Sistema	• Insertar los repositorios a compilar. • Seleccionar los paquetes a compilar. • Iniciar el proceso de compilación.

Tabla 2.1: Actor del Sistema

2.2. Listado de Requerimientos

Requisitos Funcionales

A continuación se listan los casos de uso con los requisitos funcionales que ellos implementan.

CU1. Importar Repositorio.

1. Insertar Repositorios.

CU2. Gestionar Paquetes.

2. Insertar en la Base de Datos.
3. Seleccionar Paquetes.

CU3. Compilar Paquetes.

4. Compilar Paquetes.
5. Manejar Errores de Compilación.
6. Subir paquete compilado a un FTP.

CU4. Mostrar Reporte.

7. Mostrar Datos de Compilación.

CU5. Recuperar Base de Datos.

8. Recuperar Base de Datos.

Requisitos no Funcionales

A continuación se listan los requisitos no funcionales.

Requerimiento de Usabilidad

1. Tipo de usuario final: usuario avanzado.
2. Tipo de aplicación informática: Aplicación en dos tipos de funcionamiento: en modo consola y/o integrada a la plataforma zentyal.
3. Finalidad: se emplea la aplicación para compilar repositorios de paquetes fuente de software o sólo aquellos paquetes que el usuario desee compilar de forma automatizada.

Requerimiento de Confiabilidad

4. Si se va la energía en el nodo de compilación, una vez que se conecte vuelve a enviar que está listo para compilar y la estación principal maneja esto.
5. Si se va la energía en la estación principal el proceso de compilación se para en el momento exacto, al empezar el mismo aquellos paquetes que se estaban compilando (en estado building) pasan a ready para volverlos a compilar.
6. La otra que surge es que cuando la pc donde están los datos se rompe, o el servidor de base de datos sufre daños no reparables en la base de datos del sistema, el sistema debe haber guardado hasta el momento los paquetes en estado sick y done para volver a cargar la base de datos y actualizar estos datos para que el sistema de compilación continúe por donde mismo estaba.
7. Si se cae la red en el nodo de compilación se queda esperando la conexión con la estación principal o con el FTP para enviar la información del paquete.

Requerimiento de Eficiencia

8. El sistema debe ser capaz de responder eficientemente, por tanto, cada transacción tiene un tiempo máximo de duración de un segundo.
9. La memoria RAM de la estación servidor puede tener una capacidad mínima de un 1GB.
10. La memoria RAM de la estación cliente puede tener una capacidad mínima de un 1GB.
11. Debe existir un disco duro para almacenar los paquetes compilados y este puede tener la

capacidad mínima de 80 GB.

Requerimiento de Restricciones de diseño

12. Las normas de codificación se van a llevar a cabo a partir de lo definido por la guía de estilo para el código Python PEP8.
13. El sistema debe ser distribuido.
14. Para programar se va a utilizar el lenguaje de alto nivel Python y Perl.
15. Para definir la base de datos se va a utilizar el sistema de gestión de base de datos PostgreSQL.
16. Para el desarrollo del producto se van a utilizar las herramientas Eclipse y PyDev.
17. La arquitectura se va a describir bajo el patrón arquitectónico Cliente - Servidor.
18. La prioridad a los paquetes se le asigna a través de checkbox y los que ya tienen la prioridad dada por el repositorio el checkbox debe de salir desactivado.

Requerimiento para la documentación de usuarios en línea y ayuda del sistema.

19. Se brindará un manual de usuario con la descripción de lo que hace la aplicación.

Requerimiento de Interfaz

20. Los métodos de comunicación a utilizar son: protocolo ftp para comunicar el nodo de compilación con el servidor FTP, sockets que utilizan el protocolo TCP/IP para comunicar los nodos de compilación y la estación principal.
21. Se utiliza el puerto 5007 para la comunicación por sockets y 5432 para la conexión al gestor de BD.

Requerimientos Legales, de Derecho de Autor y otros.

22. Se implementará bajo la licencia GPL.

Requerimiento de Interfaces Hardware

23. Se utiliza interfaz de red para garantizar el encaminamiento.

Requerimiento de Interfaces Software

24. Como interfaz de software se utiliza la distribución libre GNU/Linux Nova para Servidores.
25. La aplicación debe estar empaquetada en un archivo de tipo .deb.

2.3. Casos de Uso

Diagrama de Casos de Uso del Sistema.

A continuación se muestra el diagrama de casos de uso del sistema en la ilustración 2.1.

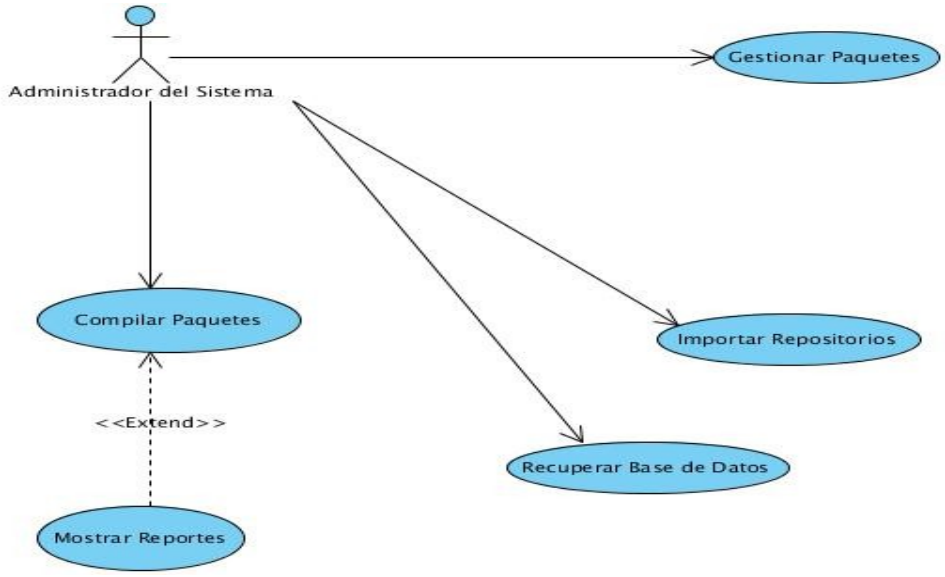


Ilustración 2.1: Diagrama de Casos de Uso

Para ver la descripción de los Casos de Uso del Sistema ver Tablas de la 1 a la 5

2.4. Diagrama de Clases del Diseño.

A continuación se explicarán cada uno de los paquetes en los que se dividió la aplicación según el patrón arquitectónico utilizado. Se muestra en la ilustración 2.2 una abstracción del diagrama de clases del diseño para una mejor comprensión de los módulos que se explican a continuación.

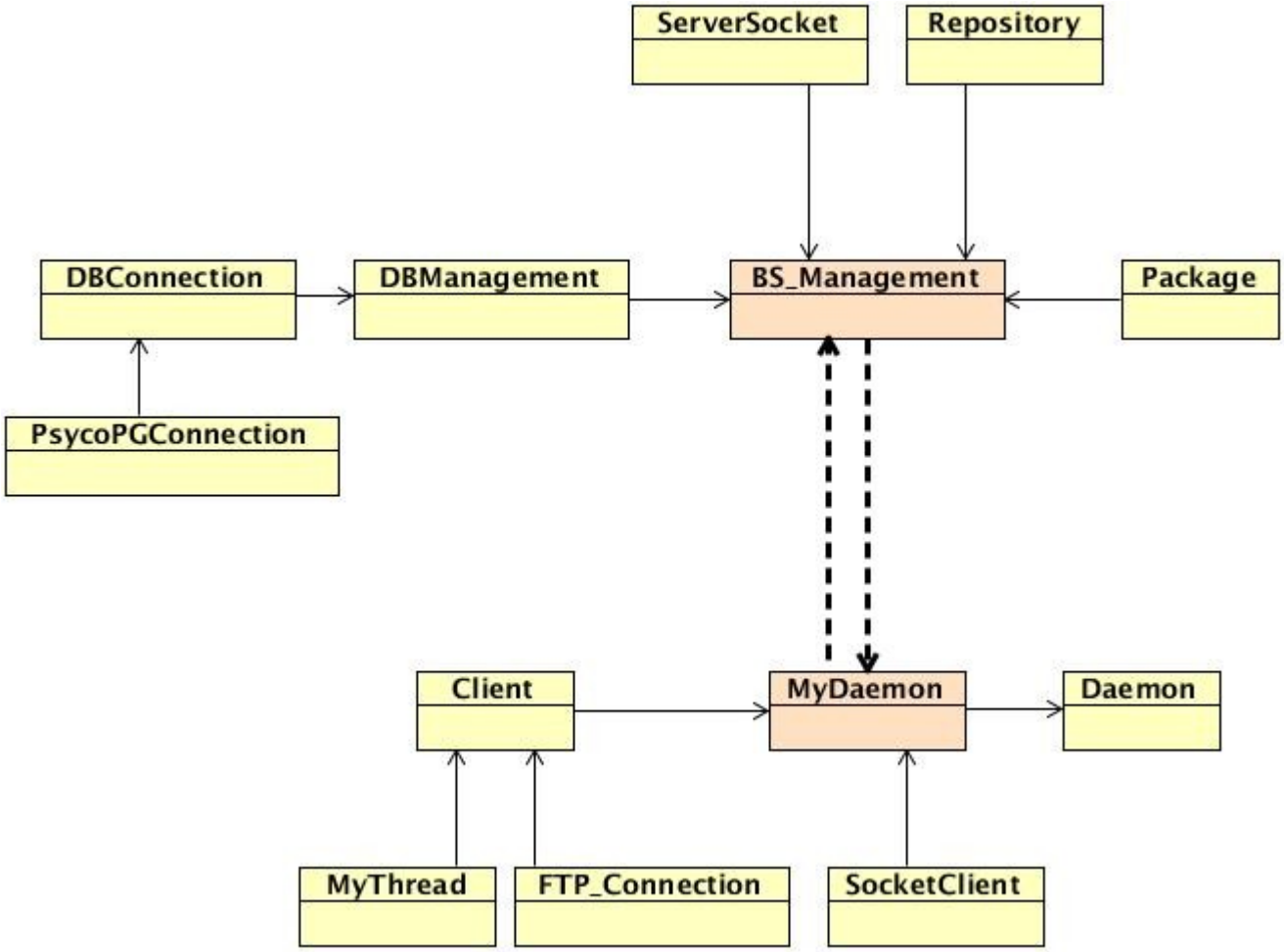


Ilustración 2.2: Abstracción del Diagrama de Clases del Diseño

Módulo db_management:

La clase DbManagement se encarga de controlar el acceso a la Base de Datos. La clase DBConecction es la clase padre, con los métodos de acceso a la Base de Datos. La clase PsycoPGConnection hereda de la clase DBConecction, es quién posee los datos y métodos para acceder a la Base de Datos usando el gestor PostgreSQL. Ver Anexo 1.

Módulo management:

La clase BSMangement se encarga de controlar todo el proceso de compilación de paquetes. La clase Package va a tener los datos que se necesitan de un paquete para realizar la compilación exitosa del mismo. La clase ServerSocket se encarga de realizar la conexión entre los nodos de compilación y la estación principal. Ver Anexo 2.

Módulo repository:

La clase Repository se encarga de almacenar y manipular los datos de un repositorio y de los elementos que lo componen. Ver Anexo 3.

Módulo ui:

La clase UI_Main_Principal se encarga de manejar la interacción entre la interfaz de la aplicación Sistema de Compilación Distribuida de Nova y todo el proceso que realiza para la compilación de paquetes. Ver Anexo 4.

Módulo client_build:

La clase Client se encarga de la descarga del paquete a compilar, compilarlo y luego subirlo al ftp. Ver Anexo 5.

Módulo connection:

La clase FTP_Connection se encarga de la comunicación con el servidor FTP que estará disponible para los paquetes una vez compilados. La clase socket_client se encarga de la comunicación con la estación principal del sistema de compilación. Ver Anexo 6.

Módulo daemon:

La clase Daemon se encarga de que el nodo de compilación corra como un demonio. La clase MyDaemon hereda de la clase Daemon, se encarga de toda la gestión de los paquetes en los mismos y de la información del servidor. Ver Anexo 7.

Módulo thread:

La clase MyThread se encarga de regular el tiempo de subida de un paquete al ftp para así saber si ocurrió algún problema de comunicación con el mismo. Ver Anexo 8.

2.5. Diagrama de Interacción

En el trabajo se definió un diagrama de colaboración por cada uno de los casos de uso. Ver Anexos del 9 al 13.

2.6. Diseño de la Base de Datos

La aplicación Sistema de Compilación Distribuida de Nova posee una sola base de datos creada en PostgreSQL. El objetivo de la base de datos es gestionar la información de todos los paquetes fuentes antes, durante y después del proceso de compilación. La manera de hacerlo es registrando todos los paquetes fuentes y las dependencias que tienen dichos fuentes; estos atraviesan por diferentes estados hasta que todos puedan ser compilados:

La estructura de la base de datos es la siguiente:

- Tabla “package”: Guarda toda la información de los paquetes fuente, el estado que tiene el paquete fuente durante el proceso de compilación y el repositorio externo del que se va a compilar.
- Tablas “active_dependence” y “nonactive_dependence”: Estas tablas guardan las llaves primarias de las dependencias que tienen. A medida que avanza el proceso de compilación la mayoría de las

dependencias quedan inutilizadas, y como forma de agilizar el trabajo con esta tabla se divide la misma en dos, trabajando mayoritariamente con la tabla “active_dependence”.

- Tabla “selfdependent”: En ocasiones ocurre que un paquete fuente depende de él mismo para compilarse, para ello se compila este paquete con una versión externa de él conocida como “instancia_autodependiente”. Esta tabla guarda el estado de la instancia autodependiente durante el proceso de compilación.
- Tabla “flag”: Durante la gestión de la información del proceso de compilación pueden ocurrir varios errores, como por ejemplo que no se haya registrado un paquete fuente que luego tiene asociado dependencias o dependientes, o que el proceso de compilación falle por alguna razón. La tabla “flag” guarda los errores que ocurren con los paquetes durante el proceso.
- Tabla “failed_msg”: Guarda los mensajes de errores que existieron en la compilación del paquete y la subida al FTP.
- Tabla “ghost_package”: guarda otros detalles de los errores que puedan ocurrir en el proceso.
- Tabla “historical_status”: Esta tabla guarda un histórico de los estados que atraviesa cada paquete. Es de gran utilidad para el administrador del sistema porque permite conocer la evolución de algún paquete y poder detectar alguna falla, o valorar por qué un paquete aún no se ha compilado, o evaluar el funcionamiento de la base de datos, u otras acciones de auditoría.
- Tabla “ip_status”: Guarda una lista de ip de los nodos de compilación que están compilando paquetes y el nombre de dichos paquetes.

Entre las funcionalidades están añadir a las tablas los nombres de los paquetes fuente disponibles y sus dependencias en los repositorios a compilar dado un esquema, ej. nova2011_i386, esto significa que los paquetes en este esquema se compilarán para nova con el nombre 2011 y arquitectura i386; generar internamente cuáles son los paquetes que están listos para compilar cuando sus dependencias están en los siguientes estados: “outside”, “sick” o “done”; asegurar los datos existentes en ella ya sea para eliminar paquetes obsoletos como para salvar los datos; actualizar el estado de los paquetes en “building”, “done” o “failed” y permite recuperar el estado de los paquetes en estado “sick” o “done”.

Como patrones de diseño utilizados en el diseño de la base de datos se utilizaron patrón de llave subrogadas y el patrón de grafo dirigido simple. Estos patrones se explican en la sección **2.7 Patrones de diseño**.

Para más información se muestra el Diagrama del Modelo de Datos.

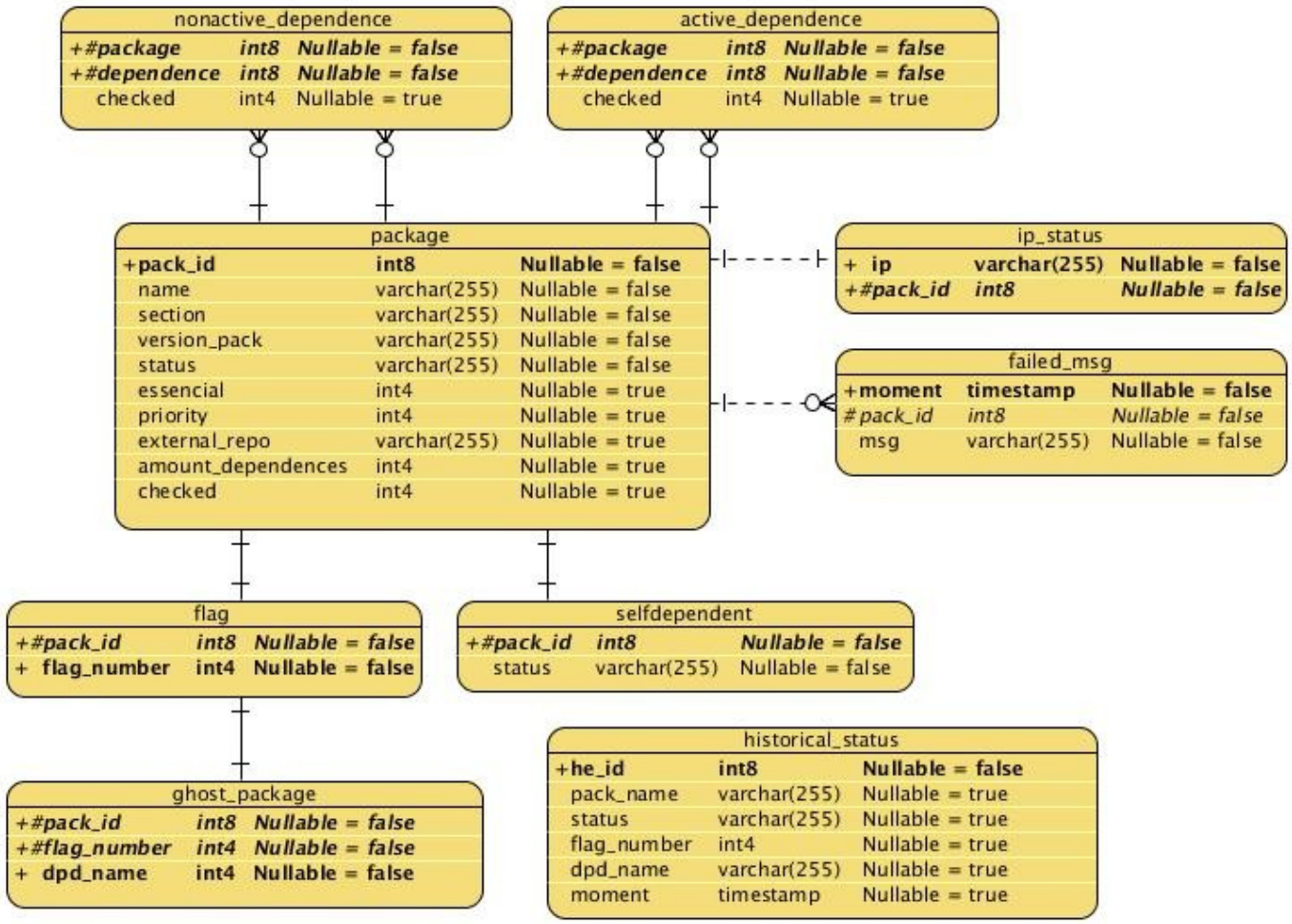


Ilustración 2.3: Modelo Físico de Datos

2.7. Patrones de diseño

Los patrones de diseño describen un problema que ocurre reiteradas veces y el núcleo de la solución al problema, de forma que pueden utilizarse en ilimitadas ocasiones sin tener que hacer dos veces lo mismo.

Los patrones de diseño son una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. En otras palabras, un patrón de diseño no es más que una solución estándar para un problema común de programación. Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios. [13.]

Patrones GRASP

GRASP es un acrónimo de General Responsibility Assigment Software Patterns (patrones generales de software para asignar responsabilidades). Son utilizados para describir los principios fundamentales del diseño y la asignación de responsabilidades. [13.]

- **Experto en Información.**

Consiste en asignar la responsabilidad a la clase que tiene la información necesaria para realizarla.

Por ejemplo, la clases BsManagement maneja toda la información relacionada con los Paquetes, datos que se guardan en la clase Package; evidenciándose así el patrón experto en información.

- **Creador**

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

Ejemplo de ellos es en la clase UI_Main_Principal que se crea una instancia de la clase BsManagement para invocar el método de inserción de paquetes a la base de datos.

bsMang = BsManagement(sch, database, source_repo, ftp_server)

bsMang.start(lista_paquetes)

- **Alta Cohesión**

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

Por ejemplo la clase ServerSocket que no hace mucho trabajo, sólo el de brindar un servidor sockets con sus respectivos métodos de enviar y recibir información.

- **Bajo Acoplamiento**

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

Este patrón se evidencia dentro de la aplicación en la capa de manejo de datos donde las clases de acceso a los datos tienen bastante independencia de las clases de abstracción de datos. Hay poca dependencia entre esas clases lo que permite una mayor reutilización.

- **DAO**

Data Access Object / Objeto de Acceso a Datos, es un patrón de diseño que centraliza todo el acceso a datos en una capa independiente, aislándolo del resto de la aplicación. Su principal beneficio es que

reduce la complejidad de los objetos de negocio al abstraerlos de la implementación real de la comunicación con la fuente de datos y de esta forma permitir una migración más fácil de fuente de datos.

El uso de este patrón en el sistema se evidencia en la clase DbManager que se encarga de realizar la gestión de los datos entre las clases que contienen las entidades y las clases controladoras.

Patrones del diseño de la Base de Datos

- **Grafo dirigido simple**

Se utiliza cuando se establece una relación de cardinalidad M:M entre instancias de una misma tabla. El patrón establece que se puede definir una relación recursiva con la tabla con cardinalidad M:M. Ver ejemplo de esto en la ilustración 2.3 en la relación de las entidades "*package*" con "*active_dependence*" y "*nonactive_dependence*".

- **Llaves subrogadas**

Plantea que se debe generar una llave primaria única para cada entidad, en vez de usar un atributo identificador en el contexto dado. Trabajar con llaves subrogadas posibilita la creación de índices de tipo btree que agilizan las búsquedas de información en la BD.

2.8. Conclusiones del Capítulo

A lo largo de este capítulo se definieron los requisitos funcionales y no funcionales para el funcionamiento de la aplicación. Se modeló el Diagrama de Casos de Uso del Sistema, el Diagrama de Clases del Diseño y los Diagramas de Interacción (colaboración) para una mejor comprensión de las funcionalidades del sistema y se definió la solución propuesta.

Capítulo 3. Implementación y Verificación de las Pruebas

El siguiente capítulo se enfoca en mostrar la descripción del diseño de las pruebas realizadas a la herramienta Sistema de Compilación Distribuida de Nova y los resultados obtenidos durante cada iteración de las pruebas. Se crearon el Diagrama de Despliegue y el Diagrama de Componentes, mostrándose sus principales componentes y sus relaciones. Se muestra además una breve explicación de los principales métodos implementados en la solución propuesta.

3.1. Diagrama de la Vista Arquitectónica

A continuación se muestra el diagrama de despliegue del sistema, en el cual se aprecia:

- **Nodo Nodos de Compilación 1 y 2**: representan ejemplos de las máquinas donde se almacenará el código fuente del paquete a compilar y el .deb del paquete compilado. Es quién sube el paquete compilado al servidor FTP e interactúa con la estación principal a través de sockets.
- **Nodo Estación Principal**: representa la estación principal de la aplicación Sistema de Compilación Distribuida de Nova, donde se manejará todo el proceso de insertado en la base de datos, el proceso de compilación y la comunicación con los nodos de compilación a través de sockets. Aquí estará montado un servidor de base de datos usando PostgreSQL.
- **Nodo Gestor de Base de Datos PostgreSQL**: Representa la conexión del sistema con la base de datos que se encuentra localmente.

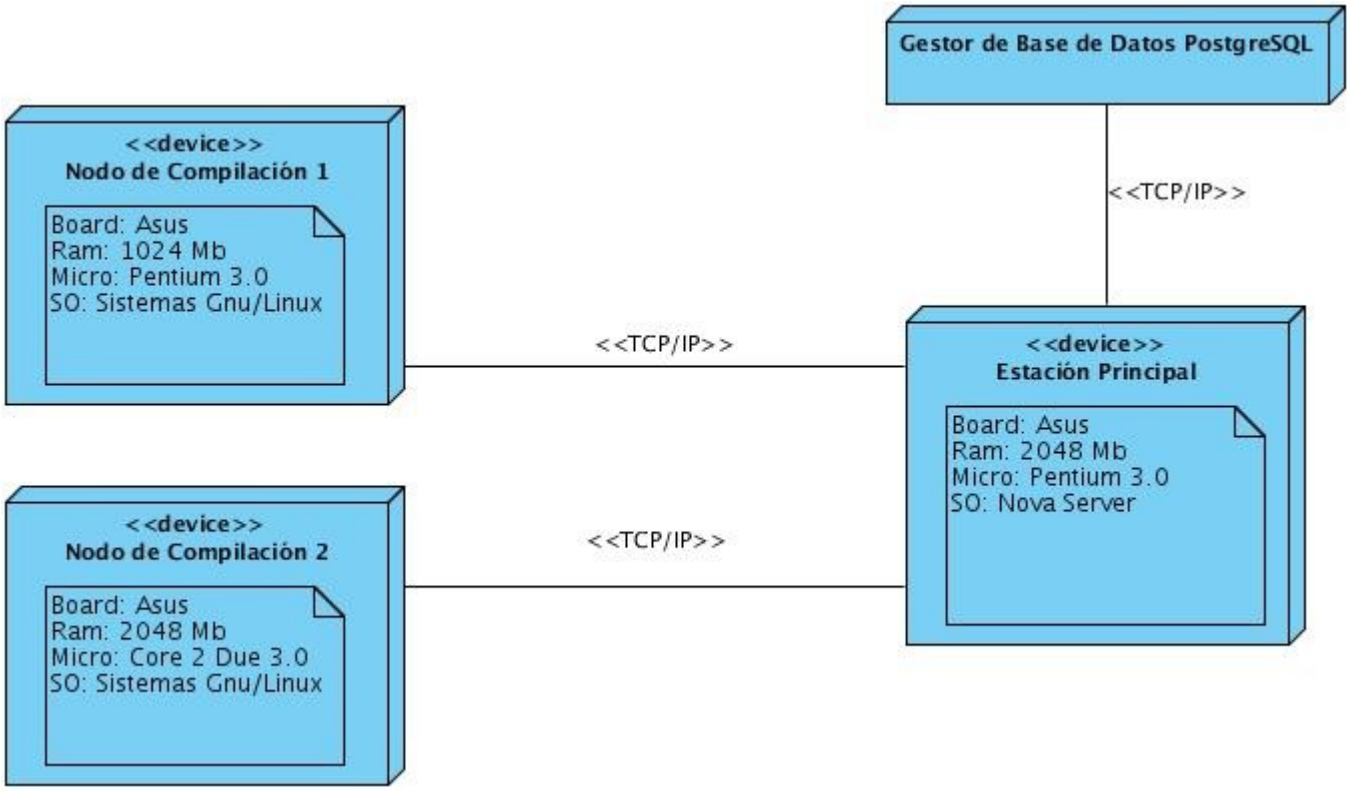


Ilustración 3.1: Diagrama de Despliegue

3.2. Diagrama de Componentes

A continuación se muestra el diagrama de componentes de los subsistemas Nodo de Compilación y Estación Principal propuesto en las ilustraciones 3.2 y 3.3 respectivamente, donde:

- **Subsistema "db_management":** contiene las clases necesarias para garantizar el acceso a datos. Las clases principales en este subsistema son "DBManagement", "PsycoPGConnection" y la librería "python-pygresql" que es quién permite manejar la base de datos desde PostgreSQL.
- **Subsistema "repository":** contiene la clase "Repository" que es quién maneja todo lo relacionado

con los repositorios a compilar, usándose las librerías “Cache” de “apt” y “SourcesList” de “aptsources”.

- **Subsistema “management”:** contiene las clases que controlan todo el proceso de acceso a datos, la comunicación con los nodos de compilación y los datos necesarios de un paquete para la compilación, usándose para esto las clases “Package”, ”BsManagement” y ”ServerSocket”.
- **Subsistema “Client_Build”:** contiene la clase “Client” que es quién se encarga de la descarga, compilación y almacenamiento del paquete en el FTP, usándose para ellos las librerías “os”, “sys” y “SourcesList” de “aptsources”.
- **Subsistema “mythread”:** contiene la clase “MyThread” que es quién se encarga del control del tiempo de subida del paquete una vez compilado al FTP, usándose las librerías “os”, “time” y “Thread”.
- **Subsistema “Connection”:** contiene las clases controladoras de las conexiones tanto a la estación servidor usándose la clases “SocketClient” así como la del FTP usándose la clase “FTP_Connection”.
- **Subsistema “Daemon”:** contiene las clases encargadas de que el nodo de compilación se ejecute como un demonio, usándose las clases “Daemon” y “MyDaemon”.
- **Clase “UI_Main_Principal”:** es la clase que interactúa con la clase “BsManagement” del subsistema management y la interfaz visual.

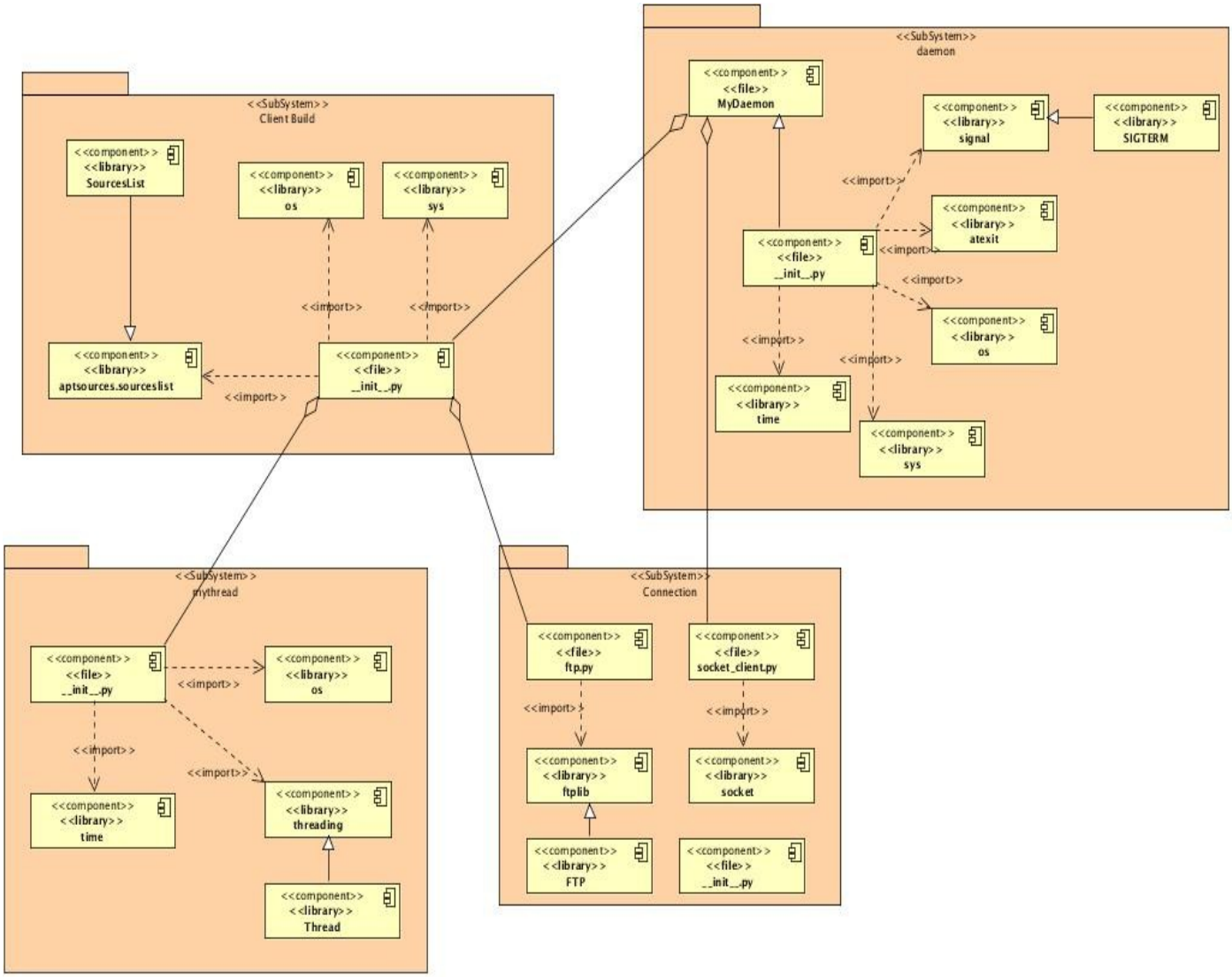


Ilustración 3.2: Diagrama de Componente Subsistema Nodo de Compilación

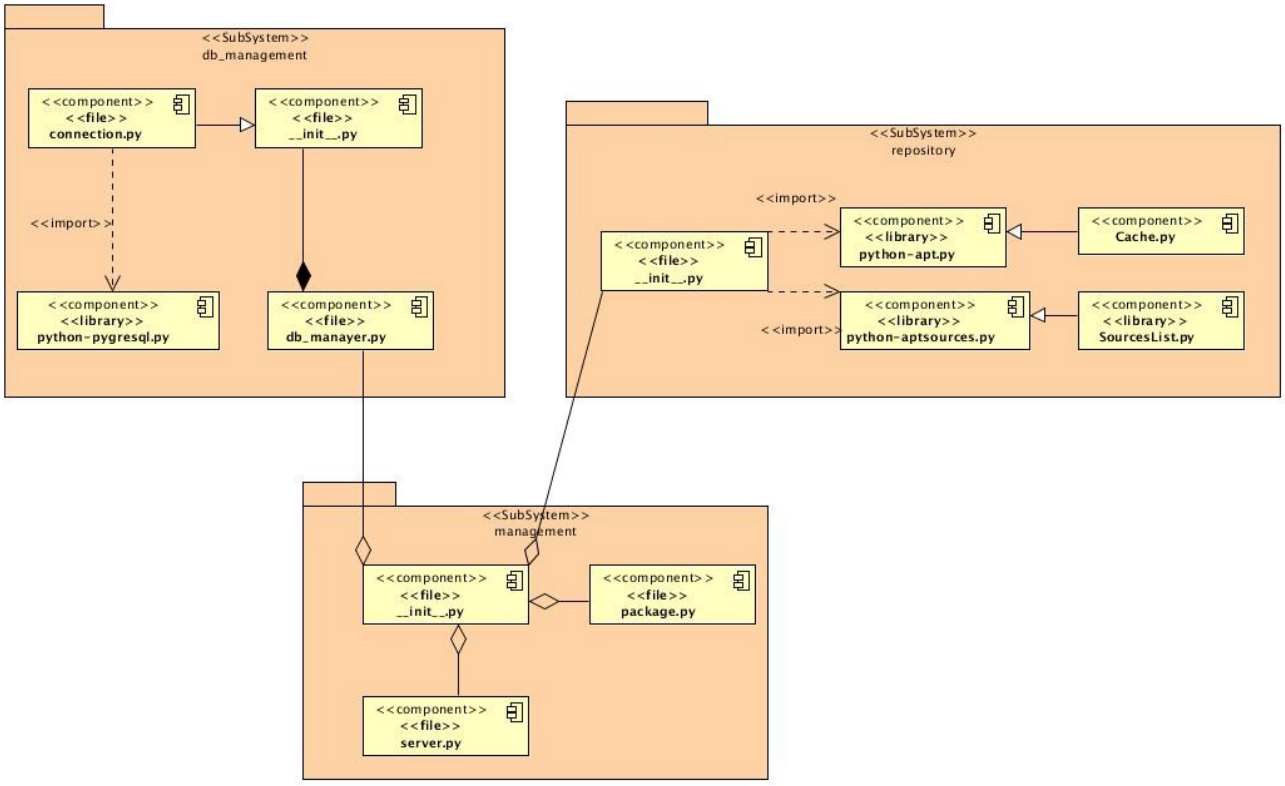


Ilustración 3.3: Diagrama de Componente del Subsistema Estación Principal

3.3. Resultados Obtenidos

Para obtener la herramienta deseada se implementaron 8 métodos principales:

- **search_enables_repositories():** para buscar los posibles repositorios fuente a compilar en la herramienta Hergire.
- **create_repositories():** para crear los repositorios que se compilarán.
- **Init_Data_Base():** para iniciar el proceso de inicialización de la base de datos.
- **Recovery_Data():** para iniciar la recuperación de la base de datos.

- **Init_Build():** para iniciar el proceso de compilación.
- **build():** encargado de compilar el paquete.
- **download_package():** encargado de descargar el paquete del repositorio.
- **upload_package():** encargado de subir el paquete al FTP.

Con la implementación de estos métodos se logra satisfacer la solución propuesta.

Método search_enables_repositories

Este método se implementa con la intención de que se conecte a la herramienta Hergire en busca de los posibles repositorios fuente que se necesiten compilar. Esto se hace a través de una conexión ftp, modificándose la lista de repositorios disponibles y devolviendo -1 si no hubo conexión o un número mayor o igual que 0 si hubo conexión, esto significa que si es 0 no se encontró ninguno para compilar y el administrador del sistema deberá insertar al menos uno manualmente y si es mayor que 0 es que sí se encontró al menos un repositorio fuente, aunque el administrador también tiene la posibilidad de añadir otro que prefiera compilar y no esté entre los encontrados. No se le pasan variables de entrada, da como salida un número y se modifica la variable “list_repos_enables”.

Método create_repositories

Este método se implementa para crear una instancia del repositorio a compilar, con el fin de obtener los paquetes fuente y sus características a través de él. Se le pasan por parámetros los repositorios fuente que el administrador del sistema decidió compilar, así como los repositorios externos que seleccionó como base para construir el repositorio binario. Se modifica la variable “repositories” que es quién posee las instancias creadas de los repositorios a compilar y se llama al método “_set_list_packages” que es quién crea las instancias de los paquetes con las características necesarias para la compilación que son: nombre del paquete, la versión, la sección, la prioridad y si es esencial o no para la compilación.

Método Init_Data_Base

Este método se implementa para iniciar la base de datos con los paquetes fuente existentes y sus dependencias, de esta manera la base de datos genera a los paquetes que se encuentran listos para compilar, que son aquellos paquetes cuyas dependencias están en el repositorio o no necesitan de ninguna dependencia para compilarse. Estos paquetes listos para compilar se les muestran al administrador del sistema para que él seleccione los que crea necesarios compilar y los que no o puede seleccionar todos y mandarlos a compilar. Se modifica la variable “readys_pack” que no es más que un diccionario de todos los paquetes listos a compilar por esquemas. Se hace instancia de la clase “BsManagement” y se llama al método “start” de la clase anteriormente mencionada pasándole por parámetros las variables “done” en True y la lista de paquetes a compilar.

Método Recovery_Data

Este método se implementa para recuperar los datos perdidos en la base de datos si ocurre un error fatal, esto es en caso extremo, ejemplo que se borre completamente la base de datos. Esto realiza en un principio los mismo pasos que el método “Init_Data_Base” pero se actualiza de ficheros de configuración que se modificaron antes del error. El código de este método es el mismo que el del método “Init_Data_Base” solo que se le pasa la variable “done” en False al método “start” de la clase BsManagement.

Método Init_Build

Este método se implementa para que le de inicio a la compilación de paquetes.

Método build

Este método se implementa para compilar el paquete. Se le pasa por parámetros el nombre del repositorio fuente, la arquitectura y el nombre del paquete a compilar. Llama al métdo “download_package” para que

descargue el paquete y poderlo compilar. Retorna True o False en dependencia de lo que se compiló.

Método download_package

Este método se implementa para que descargue el paquete del repositorio fuente. Se le pasa por parámetros el nombre del repositorio fuente y el nombre del paquete. Verifica que el paquete esté en el repositorio, en caso afirmativo descarga el paquete y retorna la dirección de dónde se guardó el fuente, en caso contrario retorna “None”.

Método upload_package

Este método se implementa para que subir el paquete luego de compilado al FTP. Se le pasa por parámetros la url, el usuario y la contraseña del FTP. Retorna True en caso de que no hubo problemas al subir el paquete y False en caso contrario.

3.4. Verificación Funcional

La calidad del software debe implementarse a lo largo de todo el ciclo de vida de un software, debe correr paralela desde la planificación del producto hasta la fase de producción de este como actividad de protección. El aspecto a considerar en el Control de la Calidad del Software es la “Prueba de Software”.

3.4.1. Pruebas de Software

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

Para un mejor acabado del producto que da respuesta a la solución propuesta se le decidió hacer ambos tipos de técnicas a la herramienta Sistema de Compilación Distribuida de Nova.

Implementación y Verificación de las Pruebas

Las técnicas de caja blanca son las pruebas que se realizan sobre las funciones internas de un módulo. Se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente.

Estas pruebas se hicieron con la biblioteca unitest de python, documentadas en el paquete test del código fuente de la aplicación. También se hicieron pruebas a la base de datos, de manera manual y arrojó errores que de manera automatizada habría sido posible detectarlos. Con estas pruebas de caja blanca se logró atravesar cada camino independiente al menos una vez.

Las técnicas de caja negra son las pruebas que se llevan a cabo sobre la interfaz del software en desarrollo. Los métodos de caja negra se centran en los requisitos funcionales del mismo e intentan encontrar errores como funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en acceso a base de datos externas, errores de rendimiento y errores de inicialización y terminación.

De la Tabla 6 a la 11 se muestran los casos de prueba basados en los requisitos, creados para iniciar con las iteraciones de pruebas que se le realizarán a la herramienta Sistema de Compilación Distribuida de Nova.

En la Tabla 12 se muestra la descripción de las variables utilizadas.

Se realizaron un total de cuatro ciclos de pruebas encontrándose un total de 27 No Conformidades, de ellas una fue un error que dio en la Base de Datos en su actualización del estado de los paquetes, 12 fueron errores del código fuente corrigiéndose los mismos en el momento en que ocurrieron y el resto fueron errores de la interfaz de consola, corrigiéndose después por los desarrolladores.

Como método de validación de la herramienta se le presentó el producto al cliente y el mismo firmó un documento donde se evidencia la conformidad con dicho producto, este documento puede ser consultado en la ilustración 14.

3.5. Conclusiones del Capítulo

A lo largo de este capítulo se realizó la descripción de los casos de prueba basados en requisitos utilizados durante las iteraciones de pruebas a la herramienta Sistema de Compilación Distribuida de Nova, así como de los resultados obtenidos durante las pruebas. Para realizar la verificación del sistema se definió utilizar las técnicas de caja blanca y caja negra, para darle un mejor acabado al producto que se deseaba obtener siguiendo los requerimientos del sistema. Se realizó la implementación de la aplicación, para ello se diseñaron los Diagramas de Componentes y de la Vista Arquitectónica para un mejor entendimiento de los componentes y relaciones de la solución propuesta.

Conclusiones

Con el estudio de las herramientas de compilación existentes a nivel mundial se comprobó que ninguna cumple con las características necesarias para las condiciones en que se encuentra la distribución cubana GNU/Linux Nova, por lo que se necesita una herramienta que automatice este proceso en dicha distribución.

Basándose en las necesidades y características del proyecto y otras exigidas por el cliente se desarrolló la aplicación Sistema de Compilación Distribuida de Nova, obteniéndose una herramienta que compila los paquetes fuente de forma distribuida y automática para la distribución cubana GNU/Linux Nova.

Para lograr la aceptación de la aplicación por el cliente se hicieron las pruebas de caja blanca, las pruebas de caja negra y las pruebas de aceptación, documentándose las mismas y sus resultados, lográndose un producto probado y aceptado por el cliente dando fe del cumplimiento de sus exigencias.

Recomendaciones

- Escoger aquellos paquetes que sean los que más paquetes dependientes de ellos posean, cuando se obtengan los paquetes listos para compilar.
- Utilizar pbuilder-uml para aprovechar las mismas funcionalidades de pbuilder en modo de usuario, sin permiso administrativo.
- El demonio en el nodo de compilación debe ejecutarse en un contexto aislado de los usuarios del sistema de forma tal que ningún usuario externo (ni siquiera un usuario con permisos administrativos) pueda modificar nada que esté enmarcado en ese contexto, usándose para esto AppArmor o SELinux.
- Para la comunicación entre los nodos de compilación y la estación principal se debe utilizar el protocolo ssh, pues en un futuro despliegue de la aplicación en la universidad se aprovechará la utilización del puerto 21 permitido por las políticas de seguridad de la entidad.

Bibliografía

- Anón. PostGreSQL. [29 abril 2012]. Disponible en la Web: <http://danielpecos.com/docs/mysql_postgres/x15.html>.
- Anón. SQLite, el motor de base de datos ágil y robusto - AplicacionesEmpresariales.com. [29 abril 2012]. Disponible en la Web: <http://www.aplicacionesempresariales.com/sqlite-el-motor-de-base-de-datos-agil-y-robusto.html>.
- Anón. [3 mayo 2012]. Disponible en la Web: <http://www.open-build-service.org/documentation/>.
- Anón. Red autocompiladora. [3 mayo 2012]. Disponible en la Web: <http://www.debian.org/devel/buildd/>.
- Jorge López. *Compilación de Paquetes*. [11 Diciembre 2011] Disponible en la Web: <http://www.iberprensa.com/todolinux/articulos/tl90_compidebian.pdf>.
- Anón. Soyuz/TechnicalDetails/Building - Launchpad Development. [9 mayo 2012]. Disponible en la Web: <https://dev.launchpad.net/Soyuz/TechnicalDetails/Building>.
- Anón. Using the Koji build system - FedoraProject. [9 mayo 2012]. Disponible en la Web: <http://fedoraproject.org/wiki/Using_the_Koji_build_system>.
- Anon. RFC 3920 – Extensible Messaging and Presence Protocol (XMPP): Core. [9 Diciembre 2011]. Disponible en la web: <http://tools.ietf.org/html/rfc3920>.
- Anon. Servidor ssh - Guía Ubuntu. [10 Diciembre 2011]. Disponible en la Web: <http://www.guia-ubuntu.org/index.php?title=Servidor_ssh>.
- Anon. ¿Que es un Socket? - Definición de Socket. [10 Diciembre 2011]. Disponible en la Web: <http://www.masadelante.com/faqs/socket>.
- SELECTING A DEVELOPMENT APPROACH. Revalidated: March 27, 2008. Retrived 27 Oct 2008.
- Anón. UML, BPMN and Database Tool for Software Development. [23 febrero 2012]. Disponible en la Web: <http://www.visual-paradigm.com/>.
- Larman, Craig. UML y Patrones. s.l. : Indiana, 2004.

Bibliografía

- Modelo de Implementación: Diagramas de Componentes y Despliegue. Ingeniería del Software (3o I.T.I.S., I.T.I.G.) Módulo 2. Tema 12: Modelo de Implementación. Febrary 21, 2012.
- Anón. MySQL: Desventajas. [20 mayo 2012]. Disponible en la Web: <http://sistemaspyt.blogspot.com/2008/09/desventajas.html>.
- Anón. MySQL: Ventajas. [20 mayo 2012]. Disponible en la Web: <http://sistemaspyt.blogspot.com/2008/09/ventajas.html>.
- Anón. Servidor genérico basado en sockets para .NET 3.5 « El Barco Tecnológico. [20 mayo 2012]. Disponible en la Web: <http://johnbarquin.wordpress.com/2010/07/25/servidor-genrico-basado-en-sockets-para-net-3-5/>.
- Colectivo de Autores. *Patrones_de_diseno_de_BD.pfd*. [UCI], mayo 2012 Disponible en la Web: <http://eva.uci.cu/file.php/180/2._Clases/Semana_2/1ra_frecuencia/MApoyo/Patrones_de_diseno_de_BD.pdf>.

Referencias Bibliográficas

1. Anon. RFC 3920 – Extensible Messaging and Presence Protocol (XMPP): Core. [9 Diciembre 2011]. Disponible en la web: <http://tools.ietf.org/html/rfc3920>.
2. Ubuntu Comunity. Instalar software - doc.ubuntu-es. [11 Diciembre 2011]. Disponible en la Web: <http://doc.ubuntu-es.org/Instalar_software>.
3. Jorge López. *Compilación de Paquetes*. [11 Diciembre 2011] Disponible en la Web: <http://www.iberprensa.com/todolinux/articulos/tl90_compidebian.pdf>.
4. Fran Berman, Anthony J.G. Hey, Geoffrey Fox (2003). *La malla informática: haciendo realidad la Infraestructura Global (Grid Computing: Making The Global Infrastructure a Reality*). Wiley. ISBN
5. Anon. Certificates – The XMPP Standards Foundation. [9 Diciembre 2011]. Disponible en la Web: <http://xmpp.org/resources/certificates/>.
6. Matthias Wimmer. [Standards-JIG] Distribution of stanza types. [9 Diciembre 2011]. Disponible en la Web: <http://mail.jabber.org/pipermail/standards/2006-May/011158.html>.
7. Matthias Wimmer. [Standards-JIG] proto-JEP: Smart Presence Distribution. [9 Diciembre 2011]. Disponible en la Web: <http://mail.jabber.org/pipermail/standards/2006-May/011182.html>.
8. Anon. Servidor ssh - Guía Ubuntu. [10 Diciembre 2011]. Disponible en la Web: <http://www.guia-ubuntu.org/index.php?title=Servidor_ssh>.
9. Anon. ¿Que es un Socket? - Definición de Socket. [10 Diciembre 2011]. Disponible en la Web: <http://www.masadelante.com/faqs/socket>.
10. Anon. Sockets en C de Unix/Linux. [10 Diciembre 2011]. Disponible en la Web: <http://www.chuidiang.com/clinux/sockets/sockets_simp.php>.
11. SELECTING A DEVELOPMENT APPROACH. Revalidated: March 27, 2008. Retrived 27 Oct 2008.
12. Anón. UML, BPMN and Database Tool for Software Development. [23 febrero 2012]. Disponible en la Web: <http://www.visual-paradigm.com/>.
13. Larman, Craig. UML y Patrones. s.l. : Indiana, 2004.
14. Anón. Guía de Usuario de Enterprise Architect 7.0. [23 abril 2012]. Disponible en la Web:

<http://www.sparxsystems.com.ar/download/ayuda/index.html?deploymentdiagram.htm>.

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

Tablas

Objetivo	Adicionar los repositorios que se utilizarán en el proceso de compilación.	
Actores	Administrador del Sistema.	
Resumen	Adicionar los repositorios que se utilizarán en el proceso de compilación.	
Complejidad	Baja	
Prioridad	Crítico	
Precondiciones	Que exista al menos un repositorio.	
Postcondiciones		
Flujo de Eventos		
Flujo Básico <Adicionar Repositorio>		
	Actor	**Sistema**
1.	Selecciona el o los repositorio(s) que se utilizarán para la compilación.	Importa el o los repositorio(s) seleccionado(s).
2.		Termina el Caso de Uso
Requisitos no Funcionales		

Tabla 1: Caso de Uso 1: Importar Repositorios

Objetivo	Selección y modificación de los paquetes que se vayan a compilar.	
Actores	Administrador del Sistema.	
Resumen	Selección y modificación de los paquetes que se vayan a compilar.	
Complejidad	Baja	
Prioridad	Critico	
Precondiciones	Que se haya seleccionado al menos un repositorio, se hayan insertado los paquetes en la BD y se hayan obtenido los paquetes listos para compilar.	
Postcondiciones		
Flujo de Eventos		
Flujo Básico <Gestionar Paquete>		
	Actor	**Sistema**
1.	Selecciona el o los paquete(s) que se compilarán.	Permite realizar varias acciones con un paquete: • Modificar paquete. Ver Sección 1: “Modificar Paquete”. • Adicionar Paquete: Ver Sección 2: “Adicionar Paquete”.
2.		Termina el Caso de Uso.
Sección 1: “Modificar Paquete”		
Flujo Básico <Modificar Paquete>		
	Actor	**Sistema**
1.	Marca el checkbox para darle o no prioridad al paquete para la compilación.	Cambia la prioridad a los paquetes.
2.		Termina el Caso de Uso.
Sección 2: “Adicionar Paquete”		
Flujo Básico <Adicionar Paquete>		
	Actor	**Sistema**
1.	Selecciona el o los paquete(s) para la compilación.	Adiciona el o los paquete(s) a una lista de compilación.
2.		Termina el Caso de Uso.
Requisitos no Funcionales	La prioridad a los paquetes se le asigna a través de checkbox y a los que ya tienen la prioridad dada por el repositorio el checkbox debe salir desactivado.	

Tabla 2: Caso de Uso 2: Gestionar Paquetes

Objetivo	Gestionar Compilación de Paquetes.		
Actores	Administrador del Sistema.		
Resumen	Gestionar Compilación de Paquetes.		
Complejidad	Alta		
Prioridad	Crítico		
Precondiciones	Que se hayan seleccionado los paquetes que se compilarán.		
Postcondiciones			
Flujo de eventos			
Flujo básico <Gestionar Compilación de Paquete>			
	Actor	**Sistema**	
1.	Inicia o Detiene el proceso de Compilación.	Permite realizar varias acciones con el proceso de Compilación: • Iniciar Compilación. Ver Sección 1: “Iniciar Compilación.” • Detener Compilación. Ver Sección 2: “Detener Compilación”.	
2.		Termina el Caso de Uso.	
3.	Decide o no ver Reporte de Paquetes Compilados.	Ver CU 4 Mostrar Reporte.	
Sección 1: “Iniciar Compilación”			
Flujo básico <Iniciar Compilación>			
	Actor	**Sistema**	
1.	Inicia el proceso de Compilación.	1.1 Envía a compilar el paquete a un nodo cliente. 1.2 Recibe el estado de la compilación. 1.3 Actualiza la Base de Datos	
2.		Termina el Caso de Uso.	
Sección 1: “Detener Compilación”			
Flujo básico <Detener Compilación>			
	Actor	**Sistema**	
1.	Detiene el proceso de compilación.	Detiene el proceso de compilación.	
2.		Termina el Caso de Uso.	
Relaciones	**CU Extendidos**	**Ver Reporte en el CU Mostrar Reporte**	
Requisitos no Funcionales			

Tabla 3: Caso de Uso 3: Compilar Paquetes

Objetivo	Reportar el estado del proceso de compilación.	
Actores	Administrador del Sistema.	
Resumen	Reportar el estado del proceso de compilación.	
Complejidad	Baja	
Prioridad	Secundario	
Precondiciones	Que se haya iniciado el proceso de compilación.	
Postcondiciones		
Flujo de eventos		
Flujo básico <Mostrar Reporte>		
	Actor	**Sistema**
1.	Solicita ver Reporte	Muestra un reporte de cuántos paquetes se intentaron compilar, cuántos se compilaron, cuántos fallaron, cuántos errores ocurrieron y cuáles fueron estos errores.
2.		Termina el Caso de Uso.
Requisitos no Funcionales		

Tabla 4: Caso de Uso 4: Mostrar Reporte

Objetivo	Recuperar los datos del proceso de compilación.	
Actores	Administrador del Sistema.	
Resumen	Recuperar los datos del proceso de compilación.	
Complejidad	Media	
Prioridad	Secundario	
Precondiciones	Que se haya iniciado un proceso de compilación.	
Postcondiciones		
Flujo de eventos		
Flujo básico <Mostrar Reporte>		
	Actor	**Sistema**
1.	Solicita recuperar datos.	1.1 Insertar en la Base de Datos los paquetes del repositorio fallido. 1.2 La Base de Datos tiene una salva de los paquetes del repositorio fallido, compara los paquetes que tenía con los que se insertaron, de no insertarse alguno de los que ya estaban se elimina. 1.3 El sistema inserta en la Base de Datos el estado de los paquetes del repositorio fallido guardados en unos ficheros como salvas que se crearon durante el proceso de compilación de ese repositorio.
2.		Termina el Caso de Uso.
Requisitos no Funcionales		

Tabla 5: Caso de Uso 5: Recuperar Base de Datos

Anexos

Ciudad de la Habana 14, junio 2012

La herramienta de compilación de los repositorios de la distribución cubana de GNU/Linux Nova denominada "Sistema de Compilación Distribuida de Nova" en su versión 0.1 cumple con los requerimientos establecidos para su desarrollo.

La presente versión:

1. Se integra con la plataforma de desarrollo de SO de Nova.
2. Genera paquetes binarios de software a partir de código fuente.
3. Gestiona colas de compilación.
4. Genera reportes de las compilaciones.
5. Presenta un módulo de recuperación ante fallos.
6. Se integra a la interfaz de administración de servidores de Nova, Zentyal.
7. Posee una interfaz CLI para la administración.

Ing. Miguel Albalat Aguila

Msc. Allan Pierra Fuentes

Ilustración 1: Acta de Aceptación del Cliente