Universidad de las Ciencias Informáticas

Facultad 1

Herramienta para la construcción de paquetes de la Distribución Cubana de GNU/Linux Nova en arquitecturas ARM y MIPS64EL

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

Autora:

Patricia Fernández Enríquez

Tutores:

Ing. Gladys Marsi Peñalver Romero

Ing. Juan Manuel Fuentes Rodríguez

Consultante:

Ing. Luis Daniel Sierra Corredera

La Habana, Junio 2017

“Año 59 de la Revolución”

Declaración de autoría

Declaro por este medio que yo Patricia Fernández Enriquez, con carné de identidad 94092030170 soy el autor principal del trabajo titulado “Herramienta para la construcción de paquetes de la Distribución Cubana de GNU/Linux Nova en arquitecturas ARM y MIPS64EL” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

_____

Patricia Fernández Enriquez

Firma del autor

_____ _____

Ing. Gladys Marsi Peñalver Romero **Ing. Juan Manuel Fuentes Rodríguez**

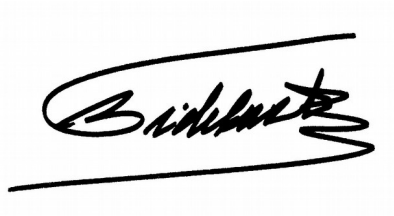
Firma del tutor Firma del tutor

_____

Ing. Luis Daniel Sierra Corredera

Firma del tutor

“El futuro de nuestra Patria tiene que ser necesariamente un futuro de hombres de ciencia, tiene que ser un futuro de hombres de pensamiento, porque precisamente es lo que más estamos sembrando; lo que más estamos sembrando son oportunidades a la inteligencia”. ”



La Habana, 15 de enero de 1960

Comandante Fidel Castro Ruz

Agradecimientos

Agradecerle a mi mamá, que a pesar de sacarla de sus casillas ha sido persistente con mis estudios y me ha apoyado; sobre todo en esta etapa final que ha sido tan difícil.

A mi tata, por ser una guía durante un tiempo porque lo superé aunque él no lo crea.

A mi novio, por ser tan dedicado y comprensivo.

A mi familia en general por su apoyo y preocupación ante el avance de este ejercicio final.

Pero agradecer sobre todas las cosas a mis tutores y mi consultante por ser tan dedicados conmigo, y aunque en ocasiones me puse molesta ellos siempre me brindaban su ayuda.

Resumen

Mediante la integración de la Universidad de las Ciencias Informáticas y la Empresa Industrial para la Informática, las Comunicaciones y la Electrónica, surge la idea de ensamblar con la Distribución Cubana de GNU/Linux Nova otros dispositivos con características de hardware diferentes a las que se conocen en la Universidad. Una de estas características viene dada por la arquitectura de procesadores *Advanced Risc Machine* y *Microprocessor without Interlocked Pipeline Stages* que presentan los nuevos dispositivos. Nova actualmente está diseñada para dar soporte a las arquitecturas i386 y AMD64. Por tal motivo es necesario una herramienta que construya el repositorio de Nova para dar soporte a estas arquitecturas de procesadores. La presente investigación propone la realización de una herramienta que construya los paquetes binarios de la Distribución Cubana de manera automática para estas arquitecturas. Para ello se realiza un estudio del arte de las herramientas que realizan la construcción de paquetes en GNU/Linux. Se define como metodología de desarrollo AUP-UCI la cual guiará el proceso de construcción de la herramienta. Se utilizan como lenguaje de programación Bash y Geany como editor de texto. Se obtuvo como resultado práctico de la investigación un script de compilación para la construcción de paquetes binarios para las arquitecturas definidas, el cual posibilita una automatización del proceso de construcción del repositorio.

Palabras clave: *arquitectura de procesadores**,** compilación, repositorio.*

Herramienta de unificación

Introducción

Las distribuciones libres del sistema GNU/Linux [1] incluyen y ofrecen únicamente software libre. Rechazan aplicaciones que no sean libres, plataformas de programación que no sean libres, controladores que no sean libres, blobs[2] de *firmware* que no sean libres, y cualquier otro tipo de software o documentación que no sea libre [1]. Poseen una base de código modular que pude ser portada[3] hacia diferentes arquitecturas con un mínimo de esfuerzo.

El software libre es un tipo de software que le permite al usuario el ejercicio de cuatro libertades básicas [2]:

1. Ejecutarlo con cualquier propósito.
2. Estudiar cómo funciona y adaptarlo a sus necesidades.
3. Distribuir copias.
4. Mejorarlo y liberar esas mejoras al público

Algunas de las ventajas que aporta este tipo de software escogido para aportar al desarrollo tecnológico en Cuba son [3]:

- Libertad de uso y redistribución.
- Independencia tecnológica.
- Fomento de la libre competencia al basarse en servicios y no licencias.
- Soporte y compatibilidad a largo plazo.

[1]Es uno de los términos empleados para referirse a la combinación del núcleo o *kernel* libre similar a Unix denominado Linux con el sistema operativo GNU.

[2]**B**inary **L**arge **Ob**jects, *objetos binarios grandes*) son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica.

[3]Portabilidad se refiere exclusivamente a la propiedad que posee un software que le permite ser ejecutado en diferentes plataformas y/o sistemas operativos.

- Formatos estándar.
- Métodos simples y unificados de gestión de software.

Cuba se encuentra en un proceso de migración a software libre de sus entidades estatales desde el primer trimestre de 2006, teniendo como misión proveer a los clientes un servicio de migración a software libre eficiente y en el menor tiempo posible con el fin de optimizar su infraestructura tecnológica con el mejor retorno de inversión [4]. Para llevar a cabo este proceso de migración la Universidad de las Ciencias Informáticas (UCI) crea diferentes proyectos productivos. Uno de estos es Servicios Integrales de Migración, Asesoría y Soporte (SIMAYS) que tiene como meta lograr la soberanía tecnológica de Cuba. Este proyecto tiene dos tareas fundamentales, brindar el servicio de migración en cualquiera de sus modalidades y desarrollar aplicaciones para apoyar el proceso de migración hacia software libre. Algunos de los proyectos realizados con éxito como parte de la migración fueron la Asesoría y acompañamiento para la migración a código abierto de la red de navegación para el Ministerio de la Construcción, permitiendo la navegación con acceso a Internet, la Migración a Código Abierto de Servicios Telemáticos para la sucursal en Pinar del Río de la Empresa Comercializadora y Exportadora DIVEP y el Servicio para la Migración a Código Abierto para la Empresa Constructora de Obras de Arquitectura e Industriales (ECOAIND).

Estos proyectos desarrollados por SYMAIS y otros que se están llevando a cabo en la actualidad están basados en 4 fundamentos conocidos como las 4s de Msc. Allan Pierra, que sumadas a las cuatro libertades del software libre garantizan la realización de productos informáticos enfocados a la migración tecnológica que se desea realizar en el país [5]:

- **Seguridad**: el modelo de desarrollo colaborativo que propone el movimiento de software libre, el acceso al código fuente y el exhaustivo proceso de revisión y auditoría de código garantizará un sistema seguro de ataques y sin puertas traseras.
- **Soberanía Tecnológica**: es la capacidad del país para desarrollarse en dicho campo en forma autónoma. No supone autarquía (independencia absoluta) sino capacidad de decidir sobre su uso y

desarrollo.

- **Socio-adaptabilidad**: las bases tecnológicas para la informatización de Cuba, deben ser hechas por cubanos y para los cubanos, logrando inigualable adaptabilidad a las condiciones de nuestro País.
- **Sostenibilidad:** la constante asimilación e investigación de las nuevas tecnologías, la planificación, los modelos novedosos de comercialización y el uso racional de los recursos humanos, materiales y naturales, garantizarán soluciones, vigencia y sostenibilidad a largo plazo.

El Centro de Software Libre (CESOL), es otro de los proyectos que apoya la migración a partir del desarrollo y actualización de la Distribución Cubana de GNU/Linux Nova, así como las acciones de capacitación y soporte que sean requeridas por los usuarios para el manejo de las aplicaciones o del propio sistema.

Nova reutiliza aplicaciones libres que se proveen para otras distribuciones de GNU/Linux, adaptándolas según las necesidades del país. Contar con el sistema propio de construcción de paquetes binarios garantiza que los binarios que se distribuyen en Nova correspondan con el código fuente que se posee. Además, trae como ventajas poder aplicar optimizaciones a nivel global a todos estos, tanto optimizaciones relacionadas con la compilación, que dan como resultados aplicaciones que consumen menos memoria y ocupan menos tiempo de CPU, como optimizaciones relacionadas con el empaquetado. Aplicar mejores algoritmos de compresión permite ahorrar espacio de almacenamiento.

El conjunto de todos los paquetes que integran una distribución se almacenan en una estructura de archivos conocida como repositorios[4], que puede estar alojado de forma local o en un servidor remoto. En ellos pueden almacenarse tanto los paquetes de código fuente comprimidos como los paquetes binarios generados a partir de estos.

La construcción del repositorio es el proceso mediante el cual un paquete fuente es procesado por un

[4] Repositorio de GNU/Linux: un repositorio de GNU/Linux es una colección de paquetes de programas de una distribución de Linux específica que generalmente contiene archivos binarios pre-compilados que pueden ser descargados e instalados por los usuarios de la distribución correspondiente. Es posible también encontrar paquetes de código fuente.

conjunto de herramientas que, siguiendo un conjunto de algoritmos definidos por el conjunto de reglas y datos establecidos por el desarrollador que empaquetó el código fuente, generan un paquete binario [6]. Para la construcción de este, se tiene en cuenta la arquitectura del microprocesador donde serán instaladas, siendo este el componente fundamental de cualquier equipo de cómputo. Las carpetas específicas de arquitectura en las fuentes del núcleo, se dirigen a detalles particulares de cada procesador soportado, mientras todas las ventajas radican en el hecho de compartir un idéntico código de núcleo [7]. De acuerdo a sus arquitecturas, existen casi tantos modelos como fabricantes de procesadores de 32 y 64 bits.

El repositorio de la Distribución Cubana de GNU/Linux Nova actualmente está construido solamente para soportar las arquitecturas de microprocesadores Intel 386, donde se desarrolló Linux originalmente y AMD64 que a pesar de ser una adaptación a los procesadores de 64 bits x86, su objetivo es soportar espacios de usuario tanto de 32 como de 64 bits en esta arquitectura.

La compañía Loongson de origen chino al realizar convenios con la empresa ensambladora de hardware de Cuba, Empresa Industrial para la Informática, las Comunicaciones y la Electrónica (GEDEME), plantea su interés en ensamblar sus computadoras con la Distribución Cubana de GNU/Linux Nova.

GEDEME, después del encuentro con los representantes chinos, ofrece al equipo de desarrollo de la Distribución Nova expandir su proyecto hacia otros dispositivos como *tablets*, *laptops*, telefonía móvil, etc. con arquitecturas como MIPS64EL de la propia compañía china y ARM una de las arquitecturas líderes de la telefonía móvil en la actualidad.

Ambas tecnologías de procesadores representan una reducción de los costos para nuestro país y como características de hardware que los procesadores ARM necesitan una cantidad menor de transistores que los procesadores x86 típicos en la mayoría de ordenadores personales y que los procesadores MIPS64el poseen hardware especialmente diseñado que les permite ejecutar código x86 estándar y tienen un acuerdo con AMD que les permite acceder al bus *HyperTransport*, así como al uso de sus chipsets (*IGPs/NorthBridge y SouthBridge* 700, 800 y 900 Series de AMD). Teniendo en cuenta lo antes descrito el

equipo de desarrollo decide llevar a cabo el proyecto propuesto por GEDEME y Loongson.

Pero construir un repositorio para otras arquitecturas sin una herramienta de construcción de paquetes lo suficientemente escalable, capaz de aprovechar al máximo el hardware disponible y que realice varias tareas de construcción de paquetes conllevaría a duplicar el esfuerzo, tiempo y recursos humanos destinados a esta labor.

Por otra parte la no existencia de esta herramienta limitaría la habilidad de manejar el crecimiento y desarrollo continuo de la distribución manteniendo la construcción del repositorio de paquetes de manera manual como se realiza en la actualidad.

Las dificultades anteriormente descritas permiten identificar el siguiente **problema de investigación***:* ¿Cómo automatizar el proceso de construcción de paquetes binarios del repositorio de la Distribución Cubana de GNU/Linux Nova para las arquitecturas ARM y MIPS64el?

Se trazó como **objeto de estudio:** el proceso de construcción de repositorios de paquetes binarios GNU/Linux.

Para el desarrollo de la investigación se concibe como **objetivo general**: desarrollar una herramienta que integre los software de compilación utilizados en la construcción de paquetes binarios en Nova, para automatizar el proceso de construcción para arquitecturas ARM y MIPS64el.

Para darle cumplimiento al objetivo general planteado, se plantean los siguientes **objetivos específicos:**

- Caracterizar el estado actual de los sistemas existentes para la construcción de paquetes binarios de las distintas distribuciones de GNU/Linux.
- Caracterizar las herramientas o tecnologías que serán usadas para la construcción de la herramienta de compilación.
- Analizar y diseñar una herramienta como solución para la construcción del repositorio de Nova en las arquitecturas MIPS64el y ARM.

Herramienta de unificación

- Implementar una herramienta de unificación para construir los paquetes binarios de la Distribución Cubana de GNU/Linux Nova para las arquitecturas ARM y MIPS64el.
- Evaluar la solución propuesta.

Se define como **campo de acción** el proceso de construcción de los repositorios de paquetes binarios de GNU/Linux Nova

Preguntas científicas

1. ¿Cuáles son las tendencias actuales relacionadas con el proceso de construcción de paquetes?
2. ¿Qué tecnologías y metodología se requieren implementar para la herramienta de unificación?
3. ¿Cómo implementar y evaluar la herramienta de unificación?

Para responder a las tareas de investigación se utilizaron los **métodos científicos:**

Métodos Empíricos

- Entrevista: en la investigación fue utilizada para obtener la información del cliente relacionado con la descripción de la herramienta. Las mismas son interpretadas como requerimientos.

Métodos Teóricos

- Histórico-lógico: empleado con el objetivo de caracterizar las soluciones existentes para el proceso de construcción de paquetes binarios a través de su evolución y desarrollo. Es utilizado para reproducir en el plano teórico el proceso de construcción de paquetes teniendo en cuenta las herramientas seleccionadas para el desarrollo de la herramienta, su funcionamiento y desarrollo.
- Analítico-Sintético: posibilita durante toda la investigación el análisis de fuentes relevantes relacionadas con las herramientas de compilación existentes, y escoger la información útil para la construcción de los repositorios de Linux. A partir de su estudio se definieron las características fundamentales para la realización del sistema.

Este documento consta de una introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía consultada, anexos y glosario de términos.

Capítulo 1. Caracterización del entorno conceptual y tecnológico para el desarrollo de la herramienta de compilación.

El presente capítulo define conceptos asociados al desarrollo de la investigación. Se realiza un análisis de las principales características y deficiencias de las herramientas de compilación existentes y finalmente, se seleccionan las herramientas que serán utilizadas para desarrollar la solución y se define la metodología a seguir durante el proceso de construcción de la herramienta de unificación.

Capítulo 2. Descripción de la solución

Se realizan las fases de inicio y ejecución de la metodología definida. Se obtiene los requisitos funcionales y no funcionales de la herramienta a desarrollar. Se explica el funcionamiento de la solución apoyada en una imagen. Ademas, se selecciona la arquitectura que sera usada para llevar a cabo la construcción de la herramienta.

Capítulo 3. Implementación y Prueba.

Se continua con la fase de ejecución de la metodología escogida y se realiza la tercera fase de esta para el desarrollo de la herramienta. Se explican los distintos tipos de pruebas, niveles y métodos que se van a utilizar para comprobar el funcionamiento de la solución. Se muestran los resultados de las pruebas y el impacto social de la investigación.

Capítulo 1. Caracterización del entorno conceptual y tecnológico para el desarrollo de la herramienta de compilación.

Para la construcción de paquetes en las distribuciones de GNU/Linux se tienen en cuenta una serie de conceptos que están definidos en este capítulo como son paquetes binarios, repositorio, dependencias de paquetes, entre otros de suma importancia para este proceso. Se lleva a cabo un estudio de las tecnologías existentes que realizan el proceso de construcción de paquetes para seleccionar de ellas, por sus características, las más convenientes para el desarrollo de la solución.

1.1 Conceptos fundamentales

1.1.1 Paquete binario

Es un paquete en el cual los componentes que contiene están listos para instalarse en el sistema operativo. En un paquete binario los archivos a instalarse están organizados siguiendo la misma estructura con que serán instalados en el sistema de archivos del sistema operativo. En el caso de las distribuciones basadas en la distribución Debian GNU/Linux, se refiere a archivos que tienen como extensión .deb o. udeb, y que cumplen con el estándar definido por los desarrolladores de la mencionada distribución [8].

1.1.2 Paquetes de código fuente

Se conoce como paquete fuente o paquete de código fuente a un paquete que contiene los elementos necesarios para construir un paquete binario. Los paquetes fuentes que contienen aplicaciones libres o de código abierto, contienen el código fuente de las mismas, así como el resto de los recursos que estas puedan requerir para su compilación, incluyendo imágenes, archivos de configuración, scripts para la creación de bases de datos. En el caso de las distribuciones basadas en Debian GNU/Linux, los paquetes fuentes conforman un grupo de archivos, uno es el que trae el empaquetado el código fuente original del programa o aplicación y el otro contiene las modificaciones o parches del desarrollador de Debian·

Además, se provee un archivo de descripción que contiene información sobre el conjunto de archivos que conforman al paquete fuente, así como firmas de verificación de integridad de los mismos, datos del mantenedor, para qué versión de la distribución de GNU/Linux están empaquetados y demás. Las extensiones que por lo general tienen estos paquetes son las siguientes: .orig.tar.bz2[5]; .diff.gzo .diff.bz2[6], .debian.tar.bz2[7] y .dsc[8]. [8]

1.1.3 Repositorio de paquetes

Un repositorio de paquetes es un lugar donde los paquetes son almacenados y mantenidos centralmente. Un repositorio puede ser local o remoto, dependiendo de si se encuentra en el sistema de archivos de la máquina del usuario que lo utiliza o si se encuentra en una localización remota accesible a través de la red. Los repositorios de paquetes generalmente cuentan con archivos índices que permiten la rápida localización de un paquete en el repositorio, así como la rápida obtención de la información del paquete sin tener que descargarlo o procesarlo. Es común encontrar que los repositorios de las distribuciones que utilizan la paquetería .deb organicen su repositorio con una estructura de árbol donde los paquetes se agrupen en directorios del sistema de archivos siguiendo un orden alfabético que facilite y optimice su búsqueda. [8]

1.1.4 Dependencias de paquetes

Las dependencias de un paquete, son aquellos paquetes binarios de los que este depende para poder funcionar en determinad entorno. Existen dos tipos de dependencias, las de tiempo de construcción o compilación (build dependency) y las de tiempo de ejecución (runtime dependency). Las dependencias de construcción son aquellos paquetes binarios que se requiere que estén instalados para poder llevar a cabo el proceso de construcción de un paquete binario a partir de un paquete fuente. Por ejemplo, el

[5] Extensión para el paquete con el código fuente original

[6] Extensión para el archivo con las modificaciones del desarrollador para versiones antiguas

[7] Extensión para el archivo con las modificaciones del desarrollador para versiones modernas

[8] Extensión para el archivo de descripción del paquete fuente

paquete que contiene al compilador del lenguaje con que esté desarrollada determinada aplicación sería una dependencia de construcción. Lo mismo pasaría con los paquetes que contienen, por ejemplo, a los archivos de encabezado (.h) no solo del núcleo de Linux, sino de la mayoría de las aplicaciones desarrolladas en C, que es casi el 85% de Linux. Por otro lado, las dependencias de tiempo de ejecución son aquellos paquetes que se requieren que estén instalados en el sistema operativo para que determinada aplicación, una vez instalada, pueda ejecutarse correctamente. Un sistema de construcción de aplicaciones o componentes de software sólo trabaja con dependencias de construcción o compilación [8].

1.1.5 Construcción de un paquete

Proceso mediante el cual un paquete fuente es procesado por un conjunto de herramientas que siguiendo un conjunto de algoritmos definidos por el conjunto de reglas y datos establecidos por el desarrollador que empaquetó el código fuente, generan un paquete binario. [9] La construcción de un paquete no siempre lleva implícito un proceso de compilación, pues a veces un paquete fuente sólo contiene datos (imágenes, tipos de letras, archivos de configuración, *scripts*) sin embargo, pueden usarse los términos construcción y compilación indistintamente al referirse a este proceso.

1.1.6 Compilación Cruzada

Proceso que ejecuta el código en otra plataforma distinta a aquella en la que el compilador se ejecuta. Es útil cuando quiere compilarse código para una plataforma a la que no se tiene acceso, o cuando es incómodo o imposible compilar en dicha plataforma (como en el caso de los sistemas embebidos) [10].

1.1.7 Sistemas embebidos

Se entiende por sistemas embebidos a una combinación de hardware y software de computadora, sumado tal vez a algunas piezas mecánicas o de otro tipo, diseñado para tener una función específica. Se pueden programar directamente en el lenguaje ensamblador del micro controlador o microprocesador

incorporado sobre el mismo, o también, utilizando los compiladores específicos [11]. Una de las ventajas de los sistemas embebidos es su flexibilidad. Ya que a la hora de realizar alguna modificación resulta mucho más sencillo modificar una línea de código al software del sistema embebido que reemplazar todo el circuito integrado.

1.2 Antecedentes de la construcción de paquetes de la Distribución Cubana de GNU/Linux Nova

Existen modos diferentes de compilar código fuente y construir paquetes para las distribuciones de GNU/Linux. A continuación, se caracterizan algunas de estas herramientas que se han usado para realizar este proceso.

1.2.1 Portage de Gentoo

En sus inicios, la Distribución Cubana de GNU/Linux Nova estaba basada en la distribución Gentoo Linux. Esta es una distribución basada en paquetes de código fuente, que permite a los usuarios optimizar al máximo su sistema a través de la compilación de sus paquetes de forma personalizada para las características específicas del hardware con que se cuente.

Gentoo es algunas veces referido como una meta-distribución debido a la extrema flexibilidad de *Portage*, lo que lo hace independiente del sistema operativo [12]. *Portage* está escrito en el lenguaje de programación *Python*, distribuido bajo los términos de la Licencia Pública General de GNU (GNU GPL), y es la utilidad principal que define a *Gentoo*. Aunque el sistema en sí es conocido como *Portage*, consiste de dos partes principales, el sistema de *ebuilds* que se ocupa del trabajo real de construir e instalar paquetes y emerge que provee una interfaz a los *ebuilds* encargada de gestionar un repositorio de *ebuilds*, resolviendo las dependencias y cuestiones similares.

Portage está caracterizado por su función principal que es compilar desde código fuente los paquetes que el usuario desea instalar. Haciendo esto permite la personalización del compilador y las opciones de la

aplicación objetivo para ajustarla a las especificaciones del sistema objetivo y los deseos específicos del usuario. Las funcionalidades relacionadas con la gestión del sistema incluyen: permitir la instalación paralela de paquete-versión, rastrear las dependencias cruzadas entre paquetes, gestionar una base de datos de paquetes instalados, proveer un repositorio de *ebuilds* locales, y sincronizar el árbol de *Portage* local con los repositorios remotos. Las funcionalidades relacionadas con paquetes individuales incluyen: la especificación de configuraciones para la máquina objetivo y escoger los componentes del paquete.

Portage es capaz de correr la compilación de varios paquetes en paralelo siempre y cuando el hardware donde se ejecute posea más de una CPU (o núcleo de CPU). La cantidad de procesos concurrentes soportados como máximo es dada por la cantidad de CPUs más uno. El proceso extra sería uno que estaría en espera para en caso de que otro se bloqueara al realizar entrada/salida, lo cual permite aprovechar al máximo el poder de cómputo con que se disponga. Sin embargo, debe esperar que las dependencias de construcción de un paquete estén instaladas antes de empezar a construirlo [13].

1.2.2 Portage con distcc

Distcc es una herramienta para acelerar la compilación de código fuente a través del uso de computación distribuida sobre una red de cómputo. Con la configuración correcta, distcc puede reducir dramáticamente el tiempo de compilación de un proyecto [14]. Está diseñado para trabajar con el lenguaje de programación C (y sus derivados, como C++ y ObjectiveC) y usa a GCC como su respaldo, aunque provee diferentes grados de compatibilidad con Intel C++ Compiler y Sun Studio Compiler Suite de Sun Microsystems.

Distribuido bajo los términos de la Licencia Pública General de GNU (GNU GPL), distcc es software libre. Distcc está diseñado para acelerar la compilación tomando ventaja del poder de procesamiento en desuso en otras computadoras. Una máquina con distcc instalado puede enviar código para ser compilado a través de la red hacia otra computadora que tenga el demonio distccd y un compilador compatible instalado [15]. Distcc trabaja como un agente para el compilador. Las máquinas remotas compilan esos

archivos fuentes sin ningunas dependencias locales (como son bibliotecas, archivos de encabezado, o definiciones de macros) a archivos objetos y los envían de regreso al origen para continuar con la fase de enlace [16]. Distcc puede trabajar de forma transparente con ccache, Portage, y Automake con una pequeña configuración [17]. Utilizar distcc disminuye el tiempo de compilación de una aplicación en específico, pero no disminuye la espera por satisfacción de dependencias.

1.2.3 Sistema de scripts para la construcción de paquetes de la Distribución Cubana de GNU/Linux Nova 2010 y 2011

A partir de Nova 2010 se cambia la base del sistema operativo y el tipo de paquetería. Empiezan a utilizarse paquetes mayormente provenientes de Ubuntu y el sistema de paquetería de Debian (.deb). Esto lleva consigo un cambio del sistema de herramientas que se usaban para la compilación y el mantenimiento de paquetes y del repositorio. Se comienzan a usar herramientas como dpkg, apt, debuild, reprepro y pbuilder. Estas herramientas permiten realizar las tareas de mantenimiento de paquetes y la construcción del repositorio. En este contexto surge un conjunto de scripts escritos en Bash que se utilizaron para mantener el repositorio y compilar paquetes. Estos fueron elaborados de una forma bastante artesanal y con una visión a corto plazo que a la larga afectó al sistema resultante, pues a medida que se fueron adicionando funcionalidades empezaron a surgir problemas de escalabilidad, dado que Bash cuenta con un lenguaje script orientada a la administración de sistemas, no de propósito general, y por tanto no cuenta con los tipos de datos que permiten implementar estructuras de datos y algoritmos eficientes [18].

Este sistema representa un retroceso en cuanto a la capacidad de compilar el repositorio de paquetes, pues sólo permite compilar un paquete a la vez, construyendo el repositorio de forma secuencial, y obligando al equipo de desarrollo a compilar sólo los paquetes que modifica mientras reutiliza los binarios provenientes de otras distribuciones de GNU/Linux.

Los sistema analizados corresponden con las alternativas mas utilizadas para la construcción de paquetes. Este estudio arrojo que los sistemas Portage Gentoo y Portage con distcc no cumplen con las características para llevar a cabo el proceso de construcción de paquetes ya que deben esperar porque las dependencias de construcción de un paquete estén instaladas antes de empezar a construirlo. Ademas, aunque el sistema de script para la construcción de paquetes de la distribución cubana de GNU/Linux Nova 2010 y 2011 realiza la construcción de paquetes unicamente de manera individual, este sistema trabaja con la paquetería .deb que permite utilizar la herramienta pbuilder la cual realiza la construcción de paquetes de manera automática y se utilizara en la herramienta de unificación a desarrollar.

1.3 Construcción avanzada de paquetes individuales en la actualidad

Existen varias herramientas que posibilitan la construcción de paquetes individuales, pero las más avanzadas son aquellas que instalan las dependencias de construcción de forma automática y lo hacen sin dañar el sistema operativo del usuario. Para lograrlo utilizan un chroot[9] en la cual se realiza un bootstrap para crear un sistema operativo base a partir del repositorio, cambiando posteriormente la raíz del sistema de archivos del proceso que está realizando la construcción del paquete con la llamada al sistema chroot antes de mandar a construir el paquete (instalando previamente sus dependencias).

1.3.1 Advanced Packaging Tool (apt)

La Herramienta Avanzada de Empaquetado (apt) es un sistema de gestión de paquetes creado por el proyecto Debian. Distribuido bajo la licencia GNU GPL y programado en C. No existe un programa APT en sí mismo, sino que es una biblioteca de funciones de C++ que se emplea por varios programas de línea de comandos para distribuir paquetes. Simplifica en gran medida la instalación y eliminación de programas en los sistemas GNU/Linux. Las herramientas APT deben ser usadas dónde se necesite mayor control de las dependencias y frecuentemente desde la línea de comandos (?consola/terminal). Tiene como ventaja

[9]Una raíz alternativa del sistema de archivos.

resolver básicamente los problemas de dependencias de paquetes y buscar los paquetes solicitados. [19] Algunos comandos para estas operaciones:

- sudo apt-get install paquete
- sudo apt-get remove paquete
- sudo apt-get -f install (Resuelve problemas con las dependencias)

1.3.2 pbuilder

Personal Package Builder o pbuilder es una herramienta distribuida bajo la licencia de GNU GPL usada por los desarrolladores de Debian y de distribuciones derivadas para la construcción de paquetes individuales en un entorno aislado mínimo dentro de un chroot. Es muy útil para determinar si un paquete se construye correctamente para una determinada distribución de GNU/Linux basada en paquetería .deb, pues al construirse sobre un sistema mínimo generado con debootstrap, permite detectar si este se empaquetó correctamente especificando todas las dependencias requeridas, algo que es muy difícil de determinar si se hace sobre un sistema operativo en producción, que puede tener dependencias instaladas previamente producto de la instalación o construcción de otros paquetes y por tanto pasar desapercibido el error provocado por no haber especificado una dependencias en los datos del paquete que se requiere probar [20].

1.3.2.1 Variantes de pbuilder

Existe una variante de pbuilder que utiliza las herramientas que provee el proyecto User Mode Linux (UML). Este proyecto provee un sistema operativo GNU/Linux corriendo en modo usuario sobre GNU/Linux. Lo que se provee con User Mode Linux es una especie de sandbox o máquina virtual donde las aplicaciones pueden ejecutarse con todos los privilegios sin peligros de afectar el sistema operativo hospedero. La variante pbuilder-user-mode-linux posibilita que el proceso de construcción de un paquete pueda ejecutarse con privilegios de superusuario tal y como si estuviera corriendo directamente en un

sistema operativo real, aunque en realidad se esté ejecutando dentro de una aplicación en modo usuario aislada del sistema operativo de la máquina [21].

1.3.3 Fakeroot y fakechroot

Existen otras variantes para ejecutar chroot, y por consiguiente debootstrap (que hace uso de chroot) sin privilegios de superusuario, utilizando herramientas como *fakeroot* y *fakechroot* que utilizan el mecanismo LD_PRELOAD del cargador de GNU/Linux para insertar bibliotecas en un ejecutable antes de que se cargue cualquier otra y así lograr sobrescribir funciones de la biblioteca libc del sistema operativo que realiza llamadas privilegiadas por otras que no requieren privilegios en un intento por simular la ejecución de estas funciones normalmente privilegiadas en espacio de usuario [22].

1.3.3 make

El programa make es una utilidad unix que facilita la descripción de dependencias entre un grupo de ficheros relacionados, generalmente todos forman parte del mismo proyecto. Los programadores utilizan make para describir cómo *hacer* un programa- qué ficheros fuente es necesario compilar, qué librerías se deben incluir y cuáles de los ficheros objeto necesitan ser enlazados. [22]. Presenta dos ventajas fundamentales:

- Es capaz de saber qué cosas hay que re compilar. Si cuando estamos depurando nuestro programa tocamos un fichero fuente, al compilar con make sólo se re compilaran aquellos ficheros que dependan del que hemos tocado. Si compilamos a mano con cc (o el compilador que sea), o tenemos en la cabeza esas dependencias para compilar sólo lo que hace falta, o lo compilamos todo. Si el proyecto es grande, se nos olvidará alguna dependencia o nos pasaremos horas compilando.
- Nos guarda los comandos de compilación con todos sus parámetros para encontrar librerías, ficheros de cabecera (.h). No tendremos que escribir largas líneas de compilación con montones de opciones que debemos saber de memoria o, al menos, sólo tendremos que hacerlo una vez.

1.3.4 dpkg

Dpkg es un programa que es la base del sistema de gestión de paquetes de Debian, desarrollado por Ian Jackson en 1993 y distribuido bajo la licencia de GNU GPL. Tiene como características ser una herramienta de bajo nivel, trabajar en conjunto con otras herramientas como apt y sirve para instalar y desinstalar paquetes. Como ventajas se tiene su utilización para instalar, quitar, y proporcionar información sobre los paquetes .deb. Para resolver conflictos con las dependencias Debian cuenta con apt. Otra desventaja que presenta es que la construcción del paquete la realiza dentro del propio sistema ocupando espacio necesario [22]. Algunos comandos para realizar los procesos de desempaquetar y construir paquetes.

- dpkg-source (Empaqueta y desempaqueta los archivos fuentes de un paquete Debian)
- dpkg-buildpackage (Es un script de control que se puede utilizar para automatizar la construcción del paquete)

1.4 Herramienta para la compilación cruzada

Luego de construir con la herramienta definida en el epígrafe anterior, el paquete debe ser compilado. Pero ante la problemática de compilarlo en otras arquitecturas debe utilizarse una herramienta que realice compilación cruzada. Estas herramientas son un conjunto de herramientas llamadas toolchains. Son útiles porque pueden compilar código para una plataforma a la que no se tiene acceso por la capacidad de almacenar las librerías de diversas arquitecturas.

1.4.1 Toolchain

Una cadena de herramientas (toolchains) es un conjunto de programas informáticos (herramientas) que se usan para crear un determinado producto (normalmente otro programa o sistema informático) [22]. Los distintos programas se suelen usar en una cadena, de modo que la salida de cada herramienta sea la

entrada de la siguiente, aunque actualmente se abusa del término para referirse a cualquier tipo de herramientas de desarrollo enlazadas. La cadena de herramientas incluyendo el propio compilador de GNU(a través de los frontends gcc y g++), el ensamblador de GNU as, y el enlazador de GNU ld, se encierran en varias etapas del proceso de compilación de un programa. Cada una ellas juega un papel fundamental, a continuación describiremos cada una de las etapas y veremos cuál es el resultado de cada una [22].

1.4.1.1 Etapas

Procesado

Es la primera fase del proceso de compilación. En esta etapa son expandidas las macros definidas en los archivos de código fuente de tipo #include, #define y #ifdef. El resultado de esta fase es un código intermedio compuesto por sentencias en C o C++, en el cual son sustituidas las constantes #define, por su valor real, son incluidos los archivos indicados con la etiqueta #include [22].

Una vez que tengamos el módulo de código fuente preparado pasaremos a ejecutar la primera fase de la compilación para ello utilizaremos el siguiente comando:

$cpp holamundo.c > holamundo.i

El resultado de este paso es un módulo de código intermedio con todas las macros expandidas. Para ver dicho módulo utilizaremos alguno de los siguientes comandos:

$vim holamundo.i

$gvim holamundo.i

En la práctica el resultado de esta fase no es guardada en disco sino que se almacena en memoria y es entregada directamente a la fase siguiente del proceso. Si se desea retener esta información en disco, basta con agregar la opción '-save-temps' en el momento de generar el módulo de código intermedio. El resultado de la etapa de pre procesado es entregado a la siguiente fase.

Compilado

Segunda fase del proceso, en esta fase se toma el código pre procesado anteriormente y el mismo es traducido, sentencia a sentencia al código objeto correspondiente. Para cada lenguaje de programación es necesario un compilador distinto, capaz de reconocer su sintaxis y que pueda analizarlo de forma correcta. En el caso de C y C++ ambos compiladores pueden compartir el mismo pre procesamiento. Como particularidad GCC y G++ no obtienen durante la fase de compilado código objeto, sino un módulo intermedio en lenguaje ensamblador [22]. Con la siguiente sentencia, podemos obtener el código ensamblador del ejemplo “Hola mundo”.

$gcc -S holamundo.i

El módulo en lenguaje ensamblador obtenido se almacena en el fichero “holamundo.s”

Ensamblado

Tercera fase del proceso, el propósito del ensamblado es tomar el código ensamblador generado, traducirlo a lenguaje de máquina y generar un módulo objeto. En esta fase se obtendrán los primeros archivos visibles del proceso de compilación, que son los módulos objeto, generados a partir de los módulos de código fuente. Cuando hay llamadas en el fichero fuente de ensamblador, el ensamblador deja las direcciones de funciones externas indefinidas para que estas sean rellenadas por el enlazador. [22] El ensamblador es invocado con la siguiente orden.

$as holamundo.s -o holamundo.o

Enlazado

La fase final del proceso de compilación es el enlazado de los módulos objeto, en caso de que fueran varios, para crear un ejecutable o una DSO en dependencia de cual sea el objetivo del proyecto. En la práctica el proceso de enlazado requiere de muchas funciones externas del sistema y bibliotecas, por lo que los comandos de enlace de GCC suelen ser bastante complejos. Afortunadamente este proceso es manejado internamente por GCC [22]. Para la llamada del enlazador basta entonces con ejecutar la siguiente orden.

$gcc holamundo.o

El resultado de esta sentencia es el enlace del fichero “holamundo.o” con la biblioteca estándar de C y produce un fichero ejecutable “a.out”. Un módulo objeto en C++ puede ser enlazado con la biblioteca estándar de C++ utilizando G++ en lugar de GCC.

1.5 Arquitectura de procesadores

Para la construcción de repositorios se tiene en cuenta un componente fundamental que es el procesador. A continuación se explica su funcionamiento y se presentan características de algunos procesadores que se utilizan en el centro y algunos por los cuales se realiza esta investigación como ARM y MIPS64EL.

Un microprocesador es un cerebro que procesa información (técnicamente la parte que realiza las operaciones se llama ALU, *Arithmetic Logic Unit* o Unidad Aritmético Lógica). Algunos de los componentes que lo conforman son registros (pequeñas memorias donde se almacenan datos), buffers, cachés, unidades de proceso y ALU. Todo esto se fabrica utilizando componentes electrónicos ciertamente pequeños (las arquitecturas actuales de nuestros ordenadores utilizan transistores de 22 nanómetros, 0.000022 milímetros) [23].

1.5.1 Intel 386

Es un microprocesador CISC1 con arquitectura x86. Durante su diseño se le llamó 'P3', debido a que era el prototipo de la tercera generación x86. Fabricado y diseñado por Intel, el procesador i386 fue una evolución importante en el mundo de la línea de procesadores que se remonta al Intel 8008. El i386 añadió una arquitectura de 32 bits y una unidad de traslación de páginas, lo que hizo mucho más sencillo implementar sistemas operativos que emplearan memoria virtual.

Debido al alto grado de compatibilidad, la arquitectura del conjunto de procesadores compatibles con el i386 suele ser llamada arquitectura i386. El conjunto de instrucciones para dicha arquitectura se conoce actualmente como IA-32 [24].

La distribución oficial i386 actualmente incluye un soporte minimalista para AMD64, que consiste en un núcleo de 64 bits, una cadena de herramientas con capacidad para crear binarios de 64 bits, y el paquete amd64-libs para ejecutar binarios de 64 bits de otros proveedores con bibliotecas compartidas nativas [33]. Nova le da soporte a esa arquitectura.

1.5.2 x86-64 o Advanced Micro Devices (AMD)

Se trata de una arquitectura desarrollada por AMD (*Advanced Micro Devices*) e implementada bajo el nombre de AMD64. Está basada en la extensión del conjunto de instrucciones x86 para manejar direcciones de 64 bits. Además de una simple extensión muestra mejoras adicionales como duplicar el número y el tamaño de los registros de uso general y de instrucciones SSE[10].

El conjunto de instrucciones del AMD x86-64 (conocido posteriormente como AMD64) es una extensión directa de la arquitectura del x86 a una arquitectura de 64 bits, motivado por el hecho de que los 4GB de memoria que son direccionables directamente por una CPU de 32 bits ya no son suficiente para todas las aplicaciones [25. Es una de las arquitecturas donde la distribución cubana esta soportada.

1.5.3 Advanced RISC Machine (ARM)

Creado alrededor de los años 80 por el *Acorn Computer Group*, como el primer procesador RISC[11] de gran impacto en el mundo. Gracias a su diseño sencillo, el ARM tiene relativamente pocos componentes en el chip, por lo que no alcanza altas temperaturas y tiene bajos requerimientos de energía. Lo anterior lo ha hecho candidato perfecto para el mercado de sistemas embebidos que van desde un celular hasta una lavadora [26].

La arquitectura ARM es el conjunto de instrucciones de 32 bits más ampliamente utilizado en unidades producidas. La relativa simplicidad de los procesadores ARM los hace ideales para aplicaciones de baja rendimiento. Como resultado, se han convertido en dominante en el mercado de la electrónica móvil e

[10] **SSE** (Streaming SIMD Extensions) es una extensión al grupo de instrucciones MMX para procesadores Pentium III, introducida por Intel en febrero de 1999.

[11] Reduced instruction set computer

integrada, encarnados en microprocesadores y micro controladores pequeños, de bajo consumo y relativamente bajo coste. Los procesadores ARM son desarrollados por ARM y los titulares de licencias de ARM.

1.5.4 Microprocessor without Interlocked Pipeline Stages (MIPS)

Siglas de *Microprocessor without Interlocked Pipeline Stages,* encierra a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies. Los diseños del MIPS son utilizados en muchos sistemas embebidos. Las primeras arquitecturas MIPS fueron implementadas en 32 bits (generalmente rutas de datos y registros de 32 bits de ancho), si bien versiones posteriores fueron implementadas en 64 bits.

Existen cinco revisiones compatibles hacia atrás del conjunto de instrucciones del MIPS, llamadas MIPS I, MIPS II, MIPS III, MIPS IV y MIPS 32/64. En la última de ellas, la MIPS 32/64 *Release* 2, se define a mayores un conjunto de control de registros. La MIPS16 que añade compresión al flujo de instrucciones para hacer que los programas ocupen menos espacio (presuntamente como respuesta a la tecnología de compresión *Thumb* de la arquitectura ARM) o la reciente MIPS MT que añade funcionalidades *multithreading* similares a la tecnología HyperThreading de los procesadores Intel Pentium 4. La adaptación a MIPS se compone en realidad de dos adaptaciones, debian-mips y debian-mipsel. MIPS consolidó su sistema de licencias alrededor de dos diseños básicos, el MIPS32 de 32 bits y el MIPS64 de 64 bits [27].

1.5.5 MIPS64

Es una tecnología RISC donde todas las instrucciones son de 32 bits para facilitar su acceso y decodificación. Tiene un bus de datos de 64 bits lo que permite el acceso simultáneo a 8 bytes. Dispone de un buen bando de registro, pues tiene 32 registros para el manejo de enteros y otros 32 para aritmética en coma flotante, todos ellos de 64 bits. Se puede configurar para comportarse de las dos maneras Little endian o big endian para almacenar valores [28].

1.5.6 MIPS64EL

Loongson es una línea de procesadores (CPU) desarrollados por la Academia China de las Ciencias en el ICT (*Institute of Computing Technology*), tiene un bajo consumo en comparación con procesadores de otras arquitecturas tales como x86 de Intel y AMD. Tiene un conjunto de instrucciones MIPS64 con Arquitectura MIPS [29]. Funciona en modo Little endian.

1.6 Metodología de desarrollo de software

La metodología de desarrollo de software que conduce el proceso es Variacion de AUP para la UCI (AUP-UCI). Esta es una adaptación de AUP que se propone para la actividad productiva de la UCI donde se logra estandarizar el proceso de desarrollo de software, dando cumplimiento además a las buenas prácticas que define CMMI-DEV v1.3. Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos. Se redujo a uno la cantidad de metodologías que se usaban y de más de veinte roles en total que se definían se redujeron a once. está centrada en potenciar las relaciones interpersonales como clave del éxito. A través de su utilización se promueve el trabajo en equipo, predominando el aprendizaje de los desarrolladores y un buen clima de trabajo. Se basa en la re alimentación continua entre el cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes, la simplicidad en las soluciones implementadas y el coraje para enfrentar los cambios [30].

AUP-UCI presenta tres fases las cuales son:

Inicio: Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: En esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se

modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre: En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Para el ciclo de vida de los proyectos de la UCI se decide tener siete disciplinas. Se consideran los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y Diseño disciplinas de la metodología. Se mantiene la disciplina Implementación al igual que en AUP, en el caso de la disciplina Prueba se divide en tres disciplinas: Pruebas Internas, de Liberación y Aceptación. Las restantes tres disciplinas que presenta AUP asociadas a la parte de gestión se cubren con las las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto).

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (Casos de Uso del Negocio, Descripción de Proceso de Negocio o Modelo Conceptual) y existen tres formas de encapsular los requisitos (Casos de Uso del Sistema, Historias de usuario, Descripción de requisitos por proceso), surgen cuatro escenarios para modelar el sistema en los proyectos. Para modelar la propuesta de solución se tomo el escenario 4 dadas sus características. Es aplicable a proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una Historia de Usuario (HU) no debe poseer demasiada información.

1.8 Lenguajes de programación

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que

controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana [31].

Para la investigación se proponen el lenguaje de programación Bash. Su nombre es un acrónimo de *Bourne-Again Shell* (otro shell bourne) haciendo un juego de palabras (born-again significa renacimiento) sobre el Bourne shell (sh), que fue uno de los primeros intérpretes importantes de Unix. Este es un lenguaje de programación de consola cuya función consiste en interpretar órdenes. Es el intérprete de comandos por defecto en la mayoría de las distribuciones de GNU con Linux. La sintaxis de órdenes de Bash es un superconjunto de instrucciones basadas en la sintaxis del intérprete Bourne [32].

1.9 Editor de texto

Para la construcción de la herramienta en su mayoría se utilizó Geany, un editor de texto pequeño y ligero basado en Scintilla con características básicas de entorno de desarrollo integrado (IDE). Utiliza bibliotecas GTK para su funcionamiento. Está disponible para distintos sistemas operativos, como GNU/Linux, Mac OS X, BSD, Solaris y Microsoft Windows. Es distribuido como software libre bajo la Licencia Pública General de GNU. Algunas de las características por lo que fue escogido para el desarrollo de la solución fueron:

- Muchos tipos de archivos soportados tales como C, Java, PHP, Python, Perl, Pascal y más.
- Construir un sistema (conjunto de ejecuciones) para compilar y ejecutar el código.

Conclusiones Parciales

A lo largo de este capítulo han sido expuestos los principales puntos de interés relacionados con los conceptos y definiciones asociados a la construcción de paquetes, lo que permitió realizar un profundo análisis de las principales características presentes en el sistema de scripts para la construcción de la Distribución Cubana de GNU/Linux Nova 2010 y 2011, que a pesar de ser un retroceso en la construcción de paquetes es seleccionado por ser utilizado actualmente por el proyecto y por trabajar con herramientas

como apt, dpkg, *pbuilder* y *make*. Ademas, el estudio de las herramientas que realizan la construcción de paquetes y se concluyó que la herramienta pbuilder es la más adecuada para la propuesta de solución.
El estudio del arte realizado permitió identificar la metodología de desarrollo de software y herramientas que se utilizarán en el desarrollo de la solución que se propone, teniendo como resultado las siguientes: como metodología de desarrollo de software AUP-UCI, como lenguaje de programación se utilizó Bash para establecer la estructura y contenido de la herramienta.

Capítulo 2. Descripción de la propuesta solución

Introducción

Una vez analizadas las herramientas que serán empleadas para el desarrollo del script de compilación, se necesita una metodología que guie el proceso de construcción del mismo. En el capítulo anterior AUP-UCI fue la metodología seleccionada. La misma está dividida en tres fases inicio, ejecución y cierre. En este capítulo se llevan a cabo las fases de inicio que es donde se determina que se hace y hasta donde llega la propuesta solución y la fase ejecución que no se realiza en su totalidad porque solamente se presentan las disciplinas donde se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño.

2.1 Propuesta de Solución

La Distribución Cubana de GNU/Linux Nova actualmente construye sus repositorios de paquetes binarios sobre y para las arquitecturas I386 y AMD64. Para el desarrollo de la propuesta solución se tuvo en cuenta cómo las arquitecturas de procesadores soportadas compilan los paquetes y cómo están compuestos. La herramienta debe ser capaz de realizar una búsqueda local o remota según decida el usuario. En el caso de realizar una búsqueda local y encontrarse el o los paquetes que se desea el usuario se procede a construirlos con la herramienta pbuilder. En caso de que no se encuentren el o los paquetes que desea el usuario, se deben descargar del repositorio de la Distribución Cubana de GNU/Linux Nova los paquetes fuentes. La herramienta debe ser capaz de leer el nombre del paquete que introdujo el usuario y descargarlo. Luego de ser descargados las fuentes empieza el proceso de construcción del paquete con la herramienta pbuilder. En caso de que se detenga la construcción se verifica si el paquete presenta dependencias no instaladas que se muestran si es el caso y se ofrece la opción de compilarlas para terminar el proceso. La construcción del paquete fuente con pbuilder genera los paquetes .deb finales con las características necesarias para el funcionamiento en otras arquitecturas.

Herramienta de unificación

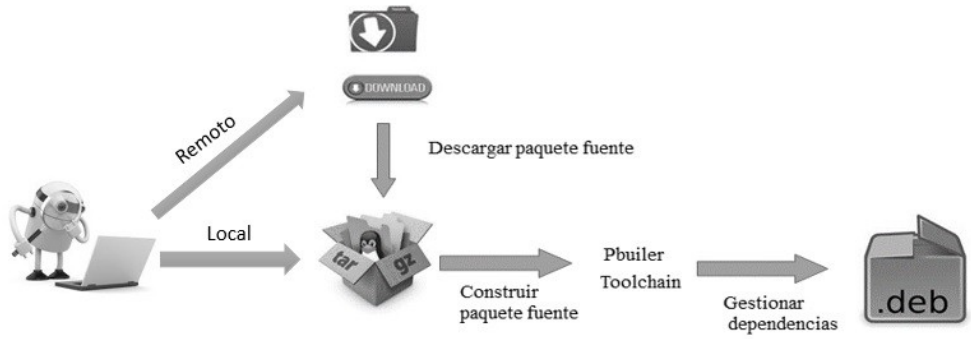


Ilustración 1: Propuesta de solución

2.2 Requisitos de la herramienta de compilación

Para el desarrollo de la solución se tienen en cuenta algunos requerimientos importantes que debe cumplir la misma. Para esto la metodología de desarrollo de software expone entre sus disciplinas la de obtención de requisito. Durante la investigación a través de la entrevista al ing. Juan Manuel Fuentes Rodríguez se obtuvieron los requisitos de la herramienta a desarrollar.

Para asegurar la correcta elección de los requisitos de la herramienta, se realizó la revisión de la documentación. Esta técnica consiste en la lectura y corrección de la completa documentación para concretar los requisitos a través del modelado y descripción del proyecto. De esta forma puedes hacer un seguimiento durante el ciclo de vida, mejorando la ingeniería de requisitos al asegurar que estos sean entregados según especificaciones.

2.2 Requisitos funcionales

Definen una función del software o sus componentes; pueden ser cálculos, detalles técnicos, manipulación

de datos y otras funcionalidades específicas que se supone, un sistema debe cumplir [33].

RF1: Buscar paquetes fuentes

RF2: Descargar un paquete fuente

RF3: Revisar sumas de verificación de la descarga

RF4: Crear un paquete individual

RF5: Mostrar dependencias de paquetes

2.3 Requisitos no funcionales(RNF)

Un requisito no funcional especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los requisitos funcionales. Sirven de apoyo a los requisitos funcionales [34].

RNF Restricciones de implementación

- Utilizar como lenguaje de programación Bash para la construcción del script en su totalidad.
- Recurrir a la herramienta pbuilder para la construcción de paquetes.

RNF Software

- Es necesario instalar el sistema operativo Nova 6.0 (2017) ya que el script está construido basado en las funcionalidades del sistema operativo.
- Debe existir un servidor de repositorio de la distribución para realizar las descargas.

2.4 Historia de Usuario

En la presente investigación se escoge la Historia de Usuario como forma de encapsular los requisitos ya que esta se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. Además presenta características semejantes a nuestro marco de

trabajo como tener el cliente siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos y ser un proyecto no muy extenso, ya que una HU no debe poseer demasiada información [35].

Número: 1	**Nombre del requisito**: Buscar los paquetes fuentes
Programador: Patricia Fernández	**Iteración Asignada:** 1
Prioridad: alta	**Tiempo Estimado:** 180 segundos
Riesgo en Desarrollo: Ausencia de desarrolladores por enfermedad. Pérdida de información imprescindible.	**Tiempo Real:** 60 segundos
Descripción: el requisito brinda la opción de buscar los paquetes que desea compilar el usuario de manera local o remota. En caso de que el paquete se encuentre local se procede a construir a partir del archivo. dsc. En caso contrario es necesario especificar el nombre del paquete que se desea construir para descargarlo del repositorio. **Escenario 1: Búsqueda local** 1. Devuelve el o los nombres de los paquetes que desea buscar. **Escenario 2: Búsqueda remota** 1. Introducir el nombre del paquete que se desea descargar. 2. Buscar en el repositorio para verificar que existe y se descarga.	
Observaciones:	
Prototipo de interfaz:	

Table 1: Historia de usuario. Buscar paquete

Herramienta de unificación

Número: 2	**Nombre del requisito:** Descargar los paquetes fuentes
Programador: Patricia Fernández	**Iteración Asignada:** 1
Prioridad: Baja	**Tiempo Estimado:** 600 segundos
Riesgo en Desarrollo: Ausencia de desarrolladores por enfermedad. Pérdida de información imprescindible.	**Tiempo Real:** 300 segundos
Descripción: El requisito brinda la opción de descargar los paquetes necesarios o que desee el usuario del repositorio. Es necesario para realizar la construcción de los paquetes para otras arquitecturas tener el archivo .dsc. Luego de la descarga se realiza una verificación para saber si esta se realizo correctamente. **Escenario 1:** 1. El usuario debe introducir el nombre del paquete que desea descargar 2. La herramienta muestra una verificación de descarga del paquete. **Escenario 2:** 1. Se puede realizar por vía ftp o http con la dirección del repositorio	
Observaciones: Esta acción se realiza si es necesario, pues el paquete puede que haya sido descargado y guardado con anterioridad.	
Prototipo de interfaz:	

Table 2: Historia de Usuario. Descargar paquete

Herramienta de unificación

Número: 3	**Nombre del requisito:** Crear paquete individual
Programador: Patricia Fernández	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 1800 segundos
Riesgo en Desarrollo: Ausencia de desarrolladores por enfermedad. Pérdida de información imprescindible	**Tiempo Real:** 600 segundos
Descripción: generar a partir de un código fuente un archivo binario que sera instalada enel sistema **Escenario 1:** 1. Crear el paquete especificado por el usuario para la arquitectura que este desee. **Escenario 2:** 2. Mostrar las dependencias no satisfechas	
Observaciones: El paquete no se construye completamente sin que las dependencias estén instaladas con anterioridad.	
Prototipo de interfaz:	

Table 3: Historia de Usuario. Crear paquete

2.5 Análisis y Diseño

Para entrar en esta disciplina ya tienen que estar definidos los requisitos y escogida la forma de encapsular los mismos. Luego dentro de la misma si se requiere, pueden ser analizados para concretar una descripción que sea fácil de mantener y ayudar a la estructuración del sistema (incluyendo su arquitectura).

La arquitectura de software siendo el diseño de más alto nivel de la estructura de un sistema encierra los requerimientos que lleva el mismo así como sus interfaces, todo esto bien relacionado y dirigido por

patrones arquitectónicos que definen la construcción y relación entre los componentes antes mencionados [36].

En la investigación se seleccionó la arquitectura Sistemas Basados en Flujos de Datos. En la misma, todo el sistema de software es visto como una serie de transformaciones en piezas consecutivas o conjunto de datos de entrada, donde los datos y las operaciones son independientes entre sí. Los datos se pueden lanzar en la topología gráfico con los ciclos, en una estructura lineal sin ciclos, o en una estructura de tipo árbol. El objetivo principal de este enfoque es lograr las cualidades de re utilización y modificación. Es adecuada para aplicaciones que involucran una serie bien definida de las transformaciones de datos independientes o cálculos en la entrada y la salida ordenada definida como compiladores.

Existen tres tipos de secuencias, Tuberías y filtros, Secuencial por lotes y Control de procesos. Para la implementación de la herramienta se selecciona el tipo Tuberías y filtros ya que este enfoque pone énfasis en la transformación gradual de los datos por componentes sucesivos. El flujo de datos es conducido por datos y todo el sistema se descompone en componentes de origen de datos, filtros, tuberías, y los sumideros de datos. Las conexiones entre los módulos son de flujo de datos que es primero en entrar / primero de salida de *buffer* que puede ser flujo de bytes, caracteres o cualquier otro tipo de este tipo. La característica principal de esta arquitectura es su ejecución concurrente [37].

Los llamados filtros no realizan forzosamente tareas de filtrado, como ser eliminación de campos o registros, sino que ejecutan formas variables de transformación, una de las cuales puede ser el filtrado. Una tubería (*pipeline*) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (*pipes*), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salida.

Las entradas de datos para las funcionalidades que se desean implementar pasan por filtros que determinan la salida o salidas que deben tener. Esto se evidencia en el ejemplo siguiente.

Un paquete trae implícito varias dependencias que resultan salidas para la compilación del paquete pero a su vez entradas que pasan por el filtro para determinar si deben ser compiladas antes de construir el paquete.

Conclusiones Parciales

En este capítulo se presentó la propuesta de solución para darle respuesta a la problemática planteada al inicio del trabajo. Se identificaron y validaron los requisitos funcionales y no funcionales que fueron posteriormente descritos en las historias de usuarios. La selección de la arquitectura tuberías y filtros permitió que la herramienta pueda ser utilizada para construcción de paquetes.

Capítulo 3. Implementación y Prueba

Introducción

Luego de conocer los requisitos que pide el cliente para desarrollar el script de compilación, se llevan a cabo algunos procedimientos para la construcción del mismo. Estos procedimientos serán vistos a lo largo del capítulo y serán probadas a través de diferentes tipos de pruebas y métodos tanto la solución propuesta en general como las funcionalidades de manera individual para lograr un producto final que cumpla con las restricciones el cliente.

3.1 Planificación de la implementación

En la planificación de la implementación se tienen en cuenta las tres historias de usuarios generadas a partir de los requisitos obtenidos. Se tendrán en cuenta para la misma una iteración y su prioridad. La característica que mayor peso tiene es la prioridad, ya que define la importancia para el desarrollo del script, se puede medir por el avance en cuanto a la implementación que defina el desarrollador. Los requisitos de prioridad alta son crear los paquetes y buscar los paquetes.

3.2 Estándar de codificación

El estándar de codificación empleado para la investigación es el Estándar de Codificación para Bash. El mismo define parámetros a seguir para llevar a cabo la implementación del script que da lugar a la investigación. Estos parámetros son de gran importancia porque contribuyen a lograr uniformidad del código y volver en un futuro el código entendible para otros desarrolladores [38]. A continuación se explican e identifican los parámetros empleados para la implementación de la solución propuesta.

Formateo del código: ofrece características que debe cumplir el código en cuanto a la identación, el tamaño máximo de línea y las líneas en blanco.

- Las líneas en blanco separan las funciones no anidadas y para indicar secciones lógicas dentro de

las mismas

```
        esac
    done
}


echo $(__ "Do you want a local search or in online repository? ")
select opt in "Local" "Repo"; do
```

Ilustración 2: Ejemplo de solución

- Las líneas tienen un límite de 79 caracteres

```
pkgs=$(cat $temp | grep -oP '(?<=Depends: ).+(?= which|but)')
```

Ilustración 3: Ejemplo de código para límite de caracteres

Codificación de caracteres: se debe utilizar UTF[12]-8 para ser reconocido ante cualquier desarrollador el significado de cada caracter.

Comentarios: ofrece información sobre archivos y funciones. El carácter # indica que es un comentario. El comentario introductorio en ficheros debe ofrecer el nombre del archivo y su contenido. El comentario de funciones ofrece nombre de la función, descripción corta y descripción de los parámetros que recibe.

Ejemplo de la solución:

 # Descargar los paquetes fuentes

Variables: el uso de variables es importante ya que guardan los datos necesarios para la implementación. Cuando el nombre es largo debe ser separado con "_" para mejorar la legibilidad.

Ejemplo de la solución:

[12] UTF **8**-bit Unicode Transformation Format) es un formato de codificación de caracteres Unicode e ISO 10646 utilizando símbolos de longitud variable.

Herramienta de unificación

DIRREPO="$1"

3.3 Pruebas de software

Las pruebas de software son una serie de actividades que se realizan con el propósito de encontrar los posibles fallos de implementación, calidad o usabilidad de un programa u ordenador, probando el comportamiento del mismo [39].

Para la investigación se define la estrategia de prueba que encierra diferentes tipos, niveles y métodos de pruebas que servirán de guía para el correcto funcionamiento de la solución. Dentro de los tipos de prueba se tiene en cuanta el tipo dinámico. Los niveles de pruebas que se tuvieron en cuenta según el desarrollo de la solución fueron unitario y de aceptación. Dentro de las pruebas unitarias se empleo el método caja blanca y la técnica utilizada fue caminos básico.

Tipo de prueba

El tipo de prueba que se emplea para la revisión de la solución son pruebas dinámicas ya que estas encierran todas aquellas pruebas que para su ejecución requieren la ejecución de la aplicación. Además las pruebas dinámicas permiten el uso de técnicas de caja negra y caja blanca con mayor amplitud. Debido a la naturaleza dinámica de la ejecución de pruebas es posible medir con mayor precisión el comportamiento de la aplicación desarrollada.

Niveles de prueba

Pruebas Unitarias

Esta prueba es aplicable a componentes representados en el modelo de implementación, para verificar que los flujos de control y de datos estén cubiertos y que ellos funcionen como se espera. Durante la prueba de unidad, la comprobación selectiva de los caminos de ejecución es una tarea esencial. Se deben diseñar casos de prueba para detectar errores debidos a cálculos incorrectos, comparaciones incorrectas o flujos de control inapropiados. Las pruebas del camino básico y de bucles son técnicas muy efectivas

para descubrir una gran cantidad de errores en los caminos. Durante el desarrollo de la herramienta se realizó esta prueba a toda la implementación y arrojo diferentes errores tanto de sintaxis como de validación.

Pruebas de Aceptación

Son realizadas principalmente por los usuarios con el apoyo del equipo del proyecto. El propósito es confirmar que el sistema está terminado, que desarrolla puntualmente las necesidades de la organización y que es aceptado por los usuarios finales. Fue realizada por el cliente y para responder ante cualquier inquietud, el desarrollador. Logró cumplir con los requisitos del cliente.

Métodos de pruebas

El método de caja blanca permite realizar pruebas al código fuente (datos y lógica). Se trabaja con entradas, salidas y el conocimiento interno. Se basan en un minucioso examen de los detalles procedimentales del código a evaluar, por lo que es necesario conocer la lógica del programa.

A pesar de que este enfoque permite diseñar pruebas que cubran una amplia variedad de casos de prueba, podría pasar por alto partes incompletas de la especificación o requisitos sin identificar; pese a garantizar la prueba exhaustiva de todos los flujos de ejecución del código analizado [40].

Técnica de diseño de las pruebas de caja blanca

La cobertura de caminos o ruta básica son pruebas que hacen que se recorran todos los posibles caminos de ejecución, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos (definición-uso de variables), comprobación de bucles (se verifican los bucles para 0,1 e interacciones, y luego para las interacciones máximas, máximas menos uno y más uno. Garantizan que se ejecute cada instrucción del programa por lo menos una vez durante la prueba. A continuación se muestra el código para crear un paquete (Ver Ejemplo de código para construir paquetes 4) que será utilizado para el flujo de los caminos básicos.

```
build () {
    temp=$(mktemp pbuilder.XXXXXX)

    pbuilder build $1 2>&1 | tee -a $temp

    pkgs=$(cat $temp | grep -oP '(?<=Depends: ).+(?= which|but)')

    if [ ! -z $pkgs ]; then
        echo ""
        echo $(__ "You must compile before the next packages: ")
        for pkg in $pkgs; do
            echo ">>> $pkg"
        done
    fi

    rm $temp
}
```

Ilustración 4: Ejemplo de código para construir paquetes

A continuación se muestra la gráfica de flujo de la solución propuesta:

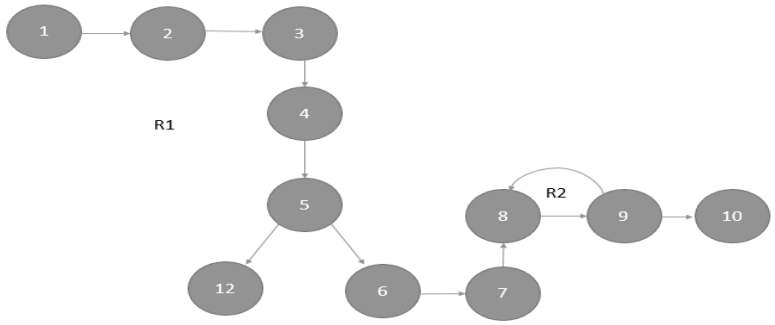


Ilustración 5: Gráfica de flujo

Considerando el grafo, encontramos el siguiente conjunto de caminos independientes:

- Camino1:1-2-3-4-5-12

- Camino2: 1-2-3-4-5-6-7-8-9-10

No se consideran caminos independientes aquellos que resulten de la combinación de otros caminos. Esta técnica se basa en la medida de complejidad ciclomática que es una métrica de software que provee una medición cuantitativa de la complejidad lógica de un programa. Define el número de caminos independientes en el conjunto básico y entrega un límite superior para el número de casos necesarios para ejecutar todas las instrucciones o al menos una.

Cálculo de la complejidad ciclomática:

- V (G)= número de regiones
- V (G)= Aristas–Nodos+2

Según el grafo:

- V (G)= 2
- V (G)=11-11+2

=2

3.4 Pruebas no funcionales

Una prueba no funcional es una prueba cuyo objetivo es la verificación de un requisito que especifica criterios que pueden usarse para juzgar la operación de un sistema (requisitos no funcionales) como por ejemplo la disponibilidad, accesibilidad, usabilidad, mantenibilidad, seguridad, rendimiento.

Para la revisión de la solución propuesta se tuvo en cuenta la **prueba de escalabilidad**, que son aquellas pruebas no funcionales que permiten determinar el grado de escalabilidad que tiene un sistema. Se entiende como escalable la capacidad que tiene el sistema para que, sin aplicar cambios drásticos en su configuración, pueda soportar el incremento de demanda en la operación. Un ejemplo que se evidencia en la solución propuesta es la modificación o incremento en la memoria RAM o CPUs en su infraestructura.

3.5 Herramienta de prueba

Herramienta de unificación

Para apoyar la revisión de la herramienta se tuvieron en cuenta los casos de prueba. Estos son un conjunto de condiciones o variables bajo las cuáles un analista determinará si una aplicación, un sistema software (software *system*), o una característica de éstos es parcial o completamente satisfactoria.

Escenario	Descripción	Variable	Respuesta del sistema	Flujo central
EC1.1 Paquete encontrado	Buscar los paquetes de manera local (en la PC) o remota en el repositorio	V Nombre del paquete que se introduzca (Debe estar escrito correctamente)	Muestra los nombres de los paquetes encontrados y en caso de buscar en el repositorio muestra un mensaje de verificación para la búsqueda	1. Abrir en la terminal del sistema 2. Ejecutar el script
EC1.2 Paquete no encontrado	Buscar los paquetes de manera local (en la PC) o remota en el repositorio	I Nombre del paquete que se introduzca (Debe estar escrito el noimbre correctamente y no se puede quedar en blanco)	Se muestra un mensaje: No se encontró ningún paquete	1. Abrir en la terminal del sistema 2. Ejecutar el script

Table 4: Caso de prueba. Encontrar paquete

Herramienta de unificación

Escenario	Descripción	Variable	Respuesta del sistema	Flujo central
EC2.1 Descarga completada	Descargar los paquetes que el usuario desee introduciendo el nombre de los mismos	V Nombre del paquete que se introduzca	Se descargan y muestran los paquetes que han sido entrados por el usuario	1. Abrir en la terminal del sistema 2. Ejecutar el script
EC2.2 Descarga no completada	Descargar los paquetes que el usuario desee introduciendo el nombre de los mismos	I Nombre del paquete que se introduzca (Debe estar escrito el noimbre correctamente y no se puede quedar en blanco)	No se descargan y se muestra el mensaje: Introduzca el nombre correctamente, el paquete no existe	1. Abrir en la terminal del sistema 2. Ejecutar el script
EC2.3 Descarga no completada	Descargar los paquetes que el usuario desee introduciendo el nombre de los mismos	I Nombre del paquete que se introduzca (Debe estar escrito el noimbre correctamente y no se puede quedar en blanco)	No se descargan y se muestra el mensaje: Debe poner el nombre del paquete que desea descargar	1. Abrir en la terminal del sistema 2. Ejecutar el script

Table 5: Caso de prueba. Descargar paquete

Herramienta de unificación

Escenario	Descripción	Variable	Respuesta del sistema	Flujo central
EC3.1 Paquete Creado	Se crea el paquete que el usuario desee para la arquitectura que desee	V Nombre del archivo .dsc Debe estar escrito el noimbre correctamente	Se muestra el .deb en la dirección que se establezca que se deba guardar	1. Abrir en la terminal del sistema 2. Ejecutar el script
EC3.2 Paquete no creado	Se crea el paquete que el usuario desee para la arquitectura que desee	I Nombre del archivo .dsc Debe estar escrito el noimbre correctamente y tiene que ser obligatoriamente esta extensión)	No se crea el paquete y muestra el mensaje: Debe crear las dependencias primero las dependencias (las que muestre el sistema)	1. Abrir en la terminal del sistema 2. Ejecutar el script

Table 6: Caso de prueba. Paquete creado

3.6 Resultados obtenidos de los casos de pruebas

Se utilizó el método caja blanca para realizar las pruebas sobre el código de la herramienta. Se encontraron 17 no conformidades en la primera iteración, 6 en la segunda y ninguna en la tercera. De estas no conformidades 6 fueron errores ortográficos, 8 de validación incorrecta y 3 de sintaxis. Con las pruebas de aceptación realizadas a nivel de cliente se pudo constatar que la herramienta cumple con los requisitos planteados.

Conclusiones Parciales

En el desarrollo del presente capítulo se especificó el uso de los estándares de codificación para lograr obtener claridad y organización en el código fuente de la solución. Se realizaron pruebas unitarias y de aceptación a la herramienta para comprobar tanto el funcionamiento del código como la calidad. Se detectaron a través de los casos de prueba 17 no conformidades que fueron satisfechas luego de realizar 3 iteraciones.

Conclusiones

Con el desarrollo de una herramienta para la construcción de paquetes para el repositorio de la Distribución Cubana de GNU/Linux Nova se da cumplimiento al objetivo general planteado. Para llegar a este resultado se concluye lo siguiente:

- Se realizó un estudio de las principales herramientas que realizan la construcción de paquetes en los sistemas GNU/Linux con el cual se pudo identificar la herramienta más adecuada para darle solución al problema de la investigación.
- Se realizó el análisis, diseño e implementación de la herramienta para la construcción de paquetes del repositorio de la Distribución Cubana de GNU/Linux Nova utilizando la herramienta pbuilder, obteniendo así una herramienta que cumple con las necesidades
- Se diseñaron y se realizaron las pruebas de software, permitiendo la correcta comprobación del funcionamiento de la solución, dando la posibilidad de realizar el proceso de entrega del software a consideración del cliente.
- Ofrece una disminución del tiempo de construcción del repositorio y de los costos de producción al requerirse menos tiempo y recursos humanos para la construcción de paquetes binarios.

Recomendaciones

Como resultado de la investigación y elementos a tener en cuenta, se hacen las siguientes recomendaciones:

- Poner en práctica la herramienta de unificación desarrollada.
- Diseñar y desarrollar una interfaz para la herramienta de unificación que facilite su usabilidad.
- Agregar a la solución otras arquitecturas de procesadores como 65xx, Altera Nios, RCA 1802 entre otras que puedan ser soportadas por la Distribución Cubana de GNU/Linux Nova.

Referencias Bibliográficas

1. Distribuciones GNU/Linux-Proyecto GNU-Free Software Fundation [citado Febrero 2017, 21] https://www.gnu.org/distros.es.html
2. https://www.gnu.org/philosophy/free-sw.es.html
3. Distribuciones GNU/Linux-Proyecto GNU-Free Software Fundation [citado Febrero 2017, 21] https://www.gnu.org/distros.es.html
4. Simays-Ecured [citado Abril 2017, 8] https://www.ecured.cu/Simays
5. PIERRA FUENTES, Allan. Nova Distribución cubana de GNU/Linux: soberanía tecnológica, seguridad, criollo. Universidad de las Ciencias Informáticas. Ciudad de la Habana. 2011. p. 9, 18, 19,29.]
6. Construcción de un repositorio de linux [citado marzo 2017, 10] https://www.google.com.cu/?gws_rd=cr,ssl&ei=l922WKC2G-X-jwTQubXoAg#q=construccion+de+un+repositorio+de+linux&*
7.]Construccion de un repositorio de linux [citado marzo 2017, 10] https://www.google.com.cu/?gws_rd=cr,ssl&ei=l922WKC2G-X-jwTQubXoAg#q=construccion+de+un+repositorio+de+linux&*
8. Fernández-Sanguino, J., 2011. The Debian GNU/Linux FAQ - Basics of the Debian package management system.Available at: http://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html
9. Compilacion cruzada linux [citado abril 2017, 23] https://www.google.com.cu/gws_rd=cr,ssl&ei=N9q1WKPLGem3jwTC77a4Cg#q=compilacion+cruzada+linux&*
10. Sistemas Embebidos (ES) – Informacion_de_referencia_ISE5_3_1.pdf[citado marzo 2017,15] http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE5_3_1.pdf
11. Gentoo Foundation, Inc., 2007. Gentoo Linux -- About Gentoo. Available at:

http://www.gentoo.org/main/en/about.xml [citado diciembre 22, 2016]

12. aurence Bonney, 2004. Reduce compile time with distcc. Available at:http://www.ibm.com/developerworks/linux/library/l-distcc/index.html
13. Robbins, D., 2004. Distcc & Distributed Computing | Dr Dobb's. Available at: http://www.drdobbs.com/distcc-distributed-computing/184401764
14. Simpson, V.L., 2004. Speed Compiling with Distcc LG #107. Available at: http://linuxgazette.net/107/simpson.html [citado diciembre 2017, 24]
15. Gentoo Foundation, Inc., 2012b. Gentoo Linux Projects -- Package Manager Specification. Available at: http://www.gentoo.org/proj/en/qa/pms.xml
16. es/Apt - Debian Wiki https://wiki.debian.org/es/Apt
17. pbuilder User's Manual https://www.netfort.gr.jp/~dancer/software/pbuilder-doc/pbuilder-doc.html
18. Dike, J., 2004. The User-mode Linux Kernel Home Page. Available at:http://user-mode-linux.sourceforge.net
19. Dassen, J.H.M., 2004. fakeroot(1). Available at: http://manpages.debian.net/cgi-bin/man.cgi?sektion=1&query=fakeroot&apropos=0&manpath=sid&locale=en
20. http://nereida.deioc.ull.es/~cleon/doctorado/doc01/herramientas/mk.html
21. http://manpages.ubuntu.com/manpages/trusty/es/man1/dpkg-source.1.html
22. Desarrollo de aplicaciones en GNU/Linux
23. Todo sobre las arquitecturas de los procesadores https://www.xataka.com/componentes/todo-obre-las-arquitecturas-de-los-procesadores
24. http://www.espaciolinux.com/foros/instalacion/que-son-las-arquitecturas-i386-i486-i586-etc-t16850.html
25. Advanced Micro Devices http://www.amd.com/Advanced%20Micro%20DevicesAMD.html
26. ARM, la 'navaja suiza' de los procesadores https://www.xataka.com/componentes/arm-la-navaja-suiza-de-los-procesadores-1

27. Linux MIPS port: mips or mipsel? | Xinotes
28. http://www.dia.eui.upm.es/asignatu/Arq_com/AC%20Grado/Paco%20Aylagas/2-MIPS64.pdf
29. https://wiki.debian.org/ArchiveQualification/mips64el#mips64el:_archive_qualification_page
30. Metodología de desarrollo para la Actividad productiva de la UCI
31. Lutz, Mark(2010). O'Reeilly Media, Inc., de. <<Learning Python, Fourth Edition>>
32. Intro_Bash/Introducción_Bash.md at master · Makova/Intro_Bash · GitHub
33. Sommerville Ian (2006). Software Engineering, 8th ed. ISBN 0-321-31379-8. (en inglés).
34. Barbosa Guerrero, Lugo Manuel. Arquitectura de software como eje temático de investigación.
35. http://mejoras.prod.uci.cu/proceso_desarrollo_produccion/workproducts/a_arquitectura_definicion_arquitectura_software_3593C399.html
36. Pressman, Roger (2002). Ingeniería del Software, un enfoque práctico
37. CESOL_Nova 6.0_Estandares de codificacion para Bash.doc
38. Pressman, Roger (2002). Ingeniería del Software, un enfoque práctico, Oc-Graw Hill
39. it-Mentor. Capacitación y guía para el desarrollo de software
40. Pressman, Roger (2002). Ingeniería del Software, un enfoque práctico, Oc-Graw Hill

Glosario de términos

- **Paquete**

Archivo comprimido utilizando algún algoritmo de compresión conocido, que puede contener los elementos que integran una aplicación u otros componentes de una distribución de GNU/Linux.

- **Compilación**

Proceso a través del cual se genera código binario, ejecutable por la CPU de una computadora, a partir del código fuente escrito en algún lenguaje de programación de alto nivel. Es decir traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.

- **Empaquetado**

Se le llama al proceso a través del cual el desarrollador de una distribución de GNU/Linux, prepara el código fuente de una aplicación, mediante un conjunto de herramientas que requieren un conjunto de archivos de reglas y datos que especifican como este se compila (en caso de que lo requiera) y se convierte en un paquete binario listo para ser instalado en la distribución o sistema operativo. El código fuente y este conjunto de archivos de reglas y datos del mantenedor son comprimidos siguiendo un formato estándar que luego las herramientas de creación de paquetes binarios son capaces de entender.

- **Debootstrap**

Herramienta que instala un sistema basado en Debian dentro de un subdirectorio de otro sistema ya instalado. Para esto no es necesario un disco (CD) de instalación, solamente acceso a al repositorio de Debian. Este puede también ser instalado y ejecutado desde otro sistema operativo. También puede ser usado para crear un rootfs para una máquina de arquitectura diferente. Esto se conoce como "cross-debootstrapping". Debootstrap solo puede usar un repositorio para sus paquetes. Si usted necesita obtener paquetes desde diferentes repositorios (de la forma en que lo hace apt) para crear un rootfs, o necesita personalizar automáticamente el rootfs, entonces utilice Multistrap. En otras palabras construye una distribución basada en paquetería .deb a partir del repositorio de paquetes.

Anexo 1

Entrevista al jefe de proyecto

¿Qué funcionalidades debe cumplir la herramienta?

¿Qué lenguaje de programación será el indicado?

¿Será necesario tener un servidor para el repositorio?

¿Qué pasa con los paquetes que no terminan de compilarse?