Universidad de las Ciencias Informáticas
Facultad 1

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

Módulo de planificación de tareas para la Herramienta de Migración y Administración de Servicios Telemáticos

Autora: Wilbia Esther Mariño Alcolea

Tutores: Ing. Miriela Velázquez Arias
Ing. Inst. Yadiel Pérez Villazón
Ing. Yosel L. Vera González

La Habana, Cuba, junio 2017
"Año 59 de la Revolución"

Declaración de Autoría

Declaro por este medio que yo Wilbia Esther Mariño Alcolea, con carné de identidad 94013046819 soy la autora principal del trabajo titulado “Módulo de planificación de tareas para la Herramienta de Migración y Administración de Servicios Telemáticos”, 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. Para que así conste firmo la presente a los ____ días del mes de _____ del año _____

Wilbia Esther Mariño Alcolea

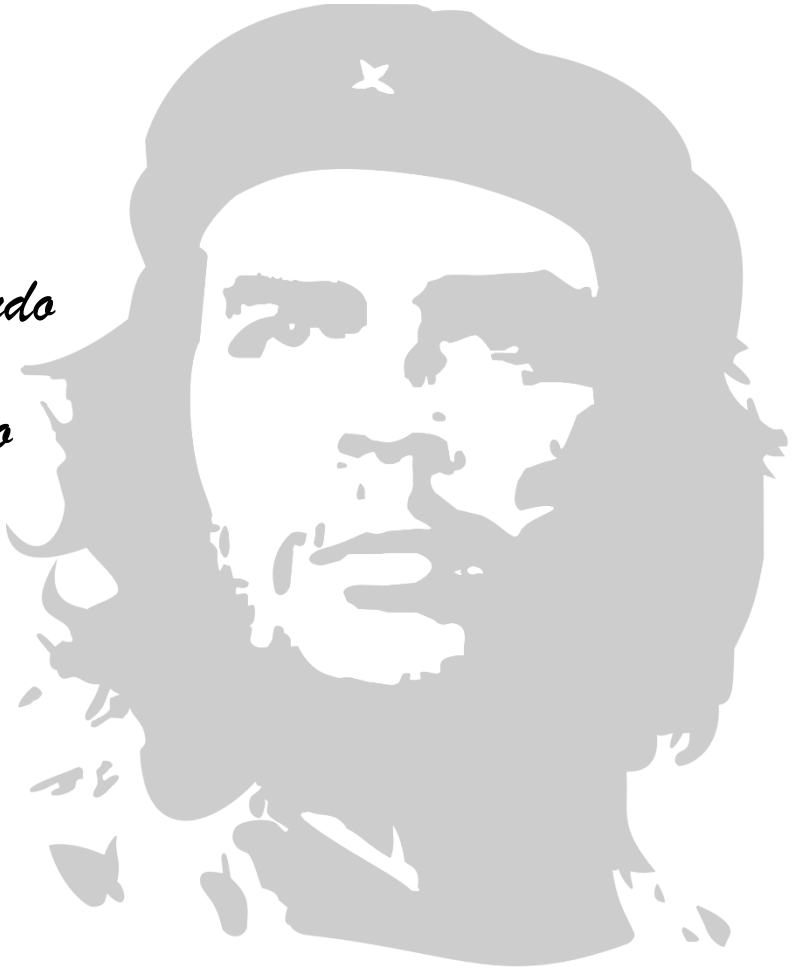
_____

Firma del autor

Ing. Miriela Velázquez Arias	Ing. Inst. Yadiel Pérez Villazón	Ing. Yosel L. Vera González
_____	_____	_____
Firma del tutor	Firma del tutor	Firma del tutor

"No se vive celebrando victorias, sino superando derrotas"

Ernesto Che Guevara



Dedicatoria

A mis padres por todo el apoyo que me han brindado a lo largo de estos duros años de la carrera, por su amor incondicional y por ser los mejores del mundo. A mis hermanos por todo su cariño.

Agradecimientos

A mis tutores por todo el apoyo que me han brindado en este período tan importante de mi vida, así como a mi compañero de tesis, Alejandro.

A mi papá que sin él no hubiese sido posible nada de lo que he logrado en toda mi vida, gracias por tu amor, tu cariño y gracias por ser el mejor padre del mundo. A mi mejor amiga y compañera, mi madre, que ha dejado todo por estar aquí conmigo en mis momentos más difíciles, te quiero ma. A mi hermano Luisito por estar molestándome cada vez que tenía un tiempo para irse de su escuela y venir aquí conmigo, tata gracias por hablarme fuerte cuando lo ameritaba el momento. A mi novio Yoel, gracias por ser mi apoyo en estos dos años. A mi tía Norma, muchas gracias mi tía por considerarme siempre tu princesa. A mis nuevos hermanitos adoptivos Ariamna y Jorge Javier (JJ), gracias por convertirse en parte de mi familia. A mis niñas de la mansión 104, Lisbet, Betsy, Raisa, Baby y Laura, gracias por estar en mis momentos más duros, algunas más que otras. Gracias a todas aquellas personas que hicieron posible que hoy pueda estar aquí.

Resumen

En el año 2004 se anuncia el acuerdo 084 tomado por el Consejo de Ministros, que orienta una migración paulatina hacia aplicaciones de código abierto. En la Universidad de las Ciencias Informáticas se encuentra el Centro de *Software* Libre, líder en los procesos de migración hacia tecnologías libres y de código abierto en Cuba. En la actualidad, dicho centro se encuentra desarrollando la Herramienta de Migración y Administración de Servicios Telemáticos, la cual debe contar con un módulo que permita la gestión de tareas programadas de forma desatendida. El presente trabajo de diploma se centró en el objetivo de desarrollar funcionalidades que permitan la planificación de tareas desde esta herramienta. Como guía de este proceso se toma la metodología de desarrollo de *software* Proceso Unificado Ágil, en la versión realizada por la Universidad de las Ciencias Informáticas, además se documentan las tecnologías a utilizar. Como resultado se obtuvo un módulo que permite realizar la administración del servicio de planificación de tareas, Cron, mediante funcionalidades que permiten gestionar tareas personalizadas y los diferentes tipos de ejecuciones a través de diferentes interfaces web.

Palabras Clave: administración, migración**,** planificación de tareas, servicios telemáticos.

Índice

Índice de Figuras

Índice de Tablas

Introducción

En los últimos años se ha hecho evidente el desarrollo de las Tecnologías de la Información y la Comunicación (TIC) debido a su impacto en diversas esferas de la vida. Específicamente los servicios telemáticos son aquellos que garantizan un flujo continuo y recíproco de información entre emisores y receptores, siendo esta una interacción de ida y vuelta, a través de una red de telecomunicaciones pública o privada. Es decir, son servicios que utilizan generalmente técnicas de procesamiento de la información de forma remota, combinando el empleo de ordenadores y redes de comunicaciones (1). Dentro de dichos servicios telemáticos se encuentra el servicio de planificación de tareas.

El servicio de planificación de tareas permite realizar, con cierta periodicidad, determinadas acciones o tareas, algunas de ellas internas del Sistema Operativo (SO), otras definidas por el administrador, e incluso algunas definidas por un usuario con los privilegios adecuados. La necesidad de este tipo de herramientas viene dada, en principio, tanto por el funcionamiento interno del sistema operativo como por la necesidad del administrador de garantizar un aceptable funcionamiento del sistema (2). En el caso de Linux, la planificación se realiza a través de Cron, que no es más que un servicio que administra los procesos que se ejecutan en segundo plano y en intervalos programados por los usuarios, resultando de gran utilidad para los usuarios que interactúan con este, debido a que favorece la gestión de tareas de forma desatendida.

La informatización de los procesos de la sociedad cubana se ha sustentado en el uso de *software* privativo, proveniente en mayor medida de empresas como *Microsoft*. El bloqueo absuelve a Cuba del pago obligatorio de productos de *software* norteamericanos empleados en el territorio nacional, a la vez que la ausencia de un mecanismo legal que muestre obligatoriedad hacia el uso de plataformas libres permite la libertad a las instituciones de usar cualquier variante de *software*, independiente del modo de licenciamiento. Ejemplo de ello es la expansión en la isla del sistema operativo *Microsoft Windows*, presente en escuelas, industrias, oficinas y otros sectores de la sociedad (3). En el año 2004 se anuncia el acuerdo 084 tomado por el Consejo de Ministros, que orienta una migración paulatina hacia aplicaciones de código abierto.

La Universidad de las Ciencias Informáticas (UCI) es una institución rectora en la producción de aplicaciones y servicios informáticos, lo cual constituye el principal soporte a la industria informática cubana. La UCI tiene como peculiaridad la vinculación del estudio con la producción. Como parte de esta infraestructura se encuentra el Centro de *Software* Libre (CESOL), líder en los procesos de migración hacia tecnologías libres y de código abierto en Cuba. Además, realiza acciones para alcanzar la independencia y soberanía tecnológicas en el país, lo cual se caracteriza por la “posibilidad de desarrollarse de forma autónoma en el campo de las

TIC, teniendo total capacidad de decisión sobre las tecnologías y la forma en que se desarrollan y usan las mismas" (4). Uno de los productos de *software* más prometedores de CESOL es la Herramienta de Migración y Administración de Servicios Telemáticos (HMAST).

HMAST permite la migración de servicios telemáticos a plataformas libres y de código abierto desde diferentes versiones de *Windows Server*, así como su posterior administración de forma remota. Esta herramienta está dirigida a apoyar el proceso masivo y progresivo de migración que se está llevando a cabo en los Organismos de la Administración Central del Estado (OACE) del país. En estos últimos resulta de vital importancia que los administradores de redes cuenten con un mecanismo que facilite la gestión de tareas programadas; dígase, por ejemplo, aquellas requeridas para el mantenimiento del sistema o actualizaciones de las aplicaciones cuando estos se encuentren ausentes.

Realizar estas acciones en sistemas GNU/Linux se torna relativamente engorroso debido a la introducción de comandos por consola y la manipulación de los archivos de configuración, lo que puede provocar en ocasiones la existencia de errores humanos, además de resultar tedioso, incurre en la pérdida de tiempo. En la actualidad se encuentran en desarrollo módulos para HMAST como correo y proxy, los cuales deben gestionar tareas programadas dando paso a la duplicación de funcionalidades implementadas y de archivos en el directorio local provocando que la administración de tareas programadas no se encuentre centralizada.

Partiendo de lo antes mencionado, se define como **problema de la investigación:** ¿Cómo garantizar la gestión de tareas programadas desde HMAST?

Se establece como **objeto de estudio** la planificación de tareas programadas en sistemas GNU/Linux; enmarcado en el **campo de acción** la planificación de tareas programadas en Nova Servidores para HMAST.

Para darle solución al problema de la investigación se determina como **objetivo general**, desarrollar funcionalidades que permitan la planificación de tareas desde HMAST. Derivándose del mismo los siguientes **objetivos específicos:**

- Analizar aspectos teóricos asociados a la investigación.
- Diseñar el sistema acoplado a la arquitectura.
- Codificar las funcionalidades definidas para el desarrollo del módulo.
- Diseñar y ejecutar pruebas al módulo a desarrollar.

Teniendo en cuenta lo planteado con anterioridad se formula la **siguiente idea a defender:** El desarrollo de un módulo para HMAST que permita la planificación de tareas programadas, proporcionará la gestión de estas de forma desatendida y la administración de tareas programadas de manera centralizada.

Se plantean las siguientes **tareas de la investigación**:

- Análisis de los principales conceptos relacionados con la investigación.
- Análisis de sistemas homólogos.
- Análisis de las características y la arquitectura de HMAST.
- Definición de requisitos funcionales y no funcionales.
- Implementación de funcionalidades.
- Aplicación de pruebas unitarias y de aceptación al módulo.

Para apoyar la presente investigación resulta necesario el uso de los siguientes **métodos científicos:**

- Histórico-lógico para la realización de un análisis de la evolución, utilización y desarrollo de los planificadores de tareas programadas.
- Analítico-sintético que permitirá el estudio de documentación, pudiéndose realizar una síntesis de los elementos fundamentales relacionados con la planificación de tareas programadas.
- Modelación para la creación de un modelo o representación de las características principales de la solución y la relación entre objetos y componentes.

El documento está estructurado en introducción, tres capítulos, conclusiones generales, referencias bibliográficas y anexos.

Capítulo 1: "Fundamentación teórica del módulo de planificación de tareas para HMAST ".

En este capítulo se dan a conocer los elementos teóricos que permiten el desarrollo de la presente investigación. Se realiza un análisis bibliográfico tomando como referencia las tareas programadas. Se describen el lenguaje de programación, las herramientas y la tecnología que serán empleadas.

Capítulo 2: "Diseño del módulo de planificación de tareas para HMAST".

En este capítulo se dan a conocer las principales características de la propuesta de solución y se definen los requisitos funcionales y no funcionales. Se muestran prototipos de interfaz de usuario, así como los patrones de diseño que se utilizarán durante la implementación.

Capítulo 3: "Implementación y pruebas del módulo de planificación de tareas para HMAST".

En este capítulo se describe la etapa de implementación de la propuesta de solución y se documentan los artefactos asociados. Se elaboran, documentan y se llevan a cabo los casos de prueba que serán aplicados.

Capítulo 1: Fundamentación teórica del módulo de planificación de tareas para HMAST

En el presente capítulo se dan a conocer los elementos teóricos que fundamentan la investigación. Se realiza un estudio de sistemas que implementan la planificación de tareas programadas, permitiendo el análisis de sus principales características. Se describe HMAST, así como las principales características que deben considerarse en el resultado final. Se da a conocer la metodología, las herramientas, las tecnologías y el lenguaje de programación que serán utilizados para el desarrollo del módulo de planificación de tareas para la HMAST.

1.1. Tareas programadas

Las tareas programadas son aquellas que el usuario puede configurar para que cualquier proceso, programa o archivo se ejecute en el tiempo o fecha que se desee (5). En GNU/Linux las tareas pueden configurarse para ejecutarse de forma automática en un período de tiempo concreto y en las fechas indicadas. Un administrador del sistema puede utilizar las tareas programadas para realizar copias de seguridad periódicas o controlar el sistema y ejecutar scripts personalizados, entre otras tareas.

1.1.1. Tareas programadas en GNU/Linux

En cualquier sistema informático se encuentran herramientas que permiten realizar, con cierta periodicidad, determinadas acciones o tareas, algunas de ellas internas del Sistema Operativo, otras definidas por el administrador, e incluso algunas definidas por un usuario con los privilegios adecuados. La necesidad de este tipo de herramientas viene dada, en principio, tanto por el funcionamiento interno del sistema operativo como por la necesidad del administrador de garantizar un aceptable funcionamiento del sistema. En los entornos GNU/Linux, los programas típicos destinados a la gestión de las tareas automatizadas son at y Cron.

Las tareas que son encomendadas a at solo se realizarán una vez. Es decir, at se utiliza para programar una tarea que se llevará a cabo en un momento determinado, y no se volverá a ejecutar (2). Este comando no es persistente, es decir que si se reinicia la PC se perderán las tareas que le son encomendadas (6).

Cron ejecuta comandos y/o programas en determinados momentos de acuerdo a lo que cada usuario ha configurado en crontab, archivo en donde se guardan las tareas programadas. Es posible ejecutar programas

en forma diaria, semanal, mensual, lo que lo hace más versátil (7). Cron es de las aplicaciones que soporta el trabajo con varios usuarios a la vez. Cada usuario puede tener su propio archivo crontab, de hecho, el etc/crontab se asume que es el archivo crontab del usuario *root*, aunque no hay problema que se incluyan otros usuarios, y de ahí que el último campo que indica cuál es el usuario que ejecuta la tarea y es obligatorio en /etc/crontab.

Anacron es el demonio que completa cron en equipos que no están encendidos todo el tiempo. Dado que generalmente las tareas recurrentes están programadas para la mitad de la noche, no ejecutarán nunca si la máquina está apagada en esos momentos. El propósito de anacron es ejecutarlas teniendo en cuenta los períodos de tiempo en los que el equipo no estuvo funcionando. Anacron frecuentemente ejecutará dichos programas unos minutos después de iniciar la máquina (8). Se utilizará el servicio Cron para el desarrollo del módulo, el mismo se describe a continuación.

1.2. Servicio de planificación de tareas

Cron es un demonio que constituye un administrador regular de procesos que se ejecuta en segundo plano y fuera del control interactivo de los usuarios. Cron es el servicio básico de los sistemas GNU/Linux usado para programar tareas, el cual siempre se encuentra en ejecución; dicho servicio asume, asimismo, que el sistema siempre está en funcionamiento (9). Ejecuta procesos o guiones a intervalos regulares, por ejemplo, cada minuto, día, semana o mes. Cron es usado a través de líneas de comando para tareas programadas, como respaldos o la ejecución de otras tareas deseadas por el usuario. En la mayoría de las distribuciones el servicio se instala automáticamente y queda iniciado desde el arranque del sistema. Cron se podría definir como el "equivalente" a Tareas Programadas de *Windows*.

La configuración del funcionamiento de Cron se encuentra dentro del directorio /etc. Los ficheros más importantes implicados en el funcionamiento del servicio Cron son:

- el propio demonio de funcionamiento: cron.d
- el fichero de configuración (disponible para root): /etc/crontab
- el fichero de inicio y parada del demonio: /etc/init.d/cron
- la orden para la programación de tareas (disponible para los usuarios con suficientes privilegios): crontab
- el sistema de informes (logs) típico de los sistemas GNU/Linux: /var/log/Cron

Para poder conocer el estado del servicio, así como iniciar o detener el mismo se deben ejecutar las órdenes correspondientes:

- Estado del demonio Cron: #/etc/init.d/Cron status o service Cron status
- Iniciar el demonio Cron: # /etc/init.d/Cron stop o service Cron stop
- Arranque del demonio Cron: # /etc/init.d/Cron start o service Cron start

De forma predeterminada, el paquete Cron incluye algunas tareas programadas que ejecutan:

- programas en el directorio /etc/Cron.hourly/ una vez por hora;
- programas en el directorio /etc/Cron.daily/ una vez por día;
- programas en el directorio /etc/Cron.weekly/ una vez por semana;
- programas en el directorio /etc/Cron.monthly/ una vez por mes (10).

Se pueden configurar varias tareas del sistema para que se hagan automáticamente. Algunas de estas tareas deben surgir en intervalos regulares, otras se deben ejecutar solo una vez, posiblemente, durante las horas de inactividad, como en la noche o durante el fin de semana. Para la programación de dichas tareas se puede utilizar Crontab.

Crontab es un archivo de texto que guarda una lista de comandos a ejecutar en un tiempo especificado por el usuario. Crontab verificará la fecha y hora en que se debe ejecutar el *scrip*t o el comando, los permisos de ejecución y lo realizará en el *background*. Cada usuario puede tener su propio archivo Crontab, de hecho, el /etc/Crontab es el archivo Crontab del usuario *root*, cuando los usuarios normales (e incluso *root*) desean generar su propio archivo de Crontab, entonces se utilizará el comando Crontab (11).

Se pueden programar tareas rutinarias de administración del sistema para que se ejecuten diariamente, semanalmente o mensualmente mediante el comando Crontab.

Entre las tareas diarias de administración del sistema Crontab se pueden incluir las siguientes:

- Eliminar archivos de pocos días de antigüedad de directorios temporales.
- Ejecutar comandos de resumen contable.
- Tomar instantáneas del sistema mediante los comandos df y ps.
- Realizar supervisiones de seguridad diaria.
- Ejecutar copias de seguridad del sistema.

Entre las tareas semanales de administración del sistema Crontab se incluye la siguiente:

- Ejecutar el comando fsck-n para mostrar problemas de disco.

Entre las tareas mensuales de administración del sistema Crontab se incluyen las siguientes:

- Mostrar archivos no utilizados durante un mes específico.
- Producir informes contables mensuales.

Además, los usuarios pueden programar comandos Crontab para ejecutar otras tareas rutinarias del sistema, como el envío de recordatorios y la eliminación de archivos de copia de seguridad. Para entender cómo se deben programar las tareas para que sean ejecutadas, es necesario entender el formato del fichero de configuración (/etc/Crontab).

1.2.1. Estructura de Crontab

Un archivo Crontab presenta la siguiente estructura:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/
# run-parts
01 * * * * root run-parts /etc/Cron.hourly
02 4 * * * root run-parts /etc/Cron.daily
22 4 * * 0 root run-parts /etc/Cron.weekly
42 4 1 * * root run-parts /etc/Cron.monthly
```

En la Tabla 1 se muestra una descripción de las cinco primeras líneas que forman parte de este archivo.

Tabla 1. Descripción de las cinco primeras líneas del archivo Crontab.

Campo	Descripción
SHELL	Intérprete de comandos
PATH	Ruta a los ejecutables
MAILTO	A quién mandar un correo si ocurre un error
HOME	Localización del directorio raíz
run-parts	Es un script que hace ejecutar los comandos del directorio especificado

A continuación de estas líneas se encuentran los campos que hacen referencia a la hora y fecha en que será ejecutada la tarea programada. Estos cinco campos que van separados por espacios se describen a continuación en la Tabla 2.

Tabla 2. Formato de las tareas.

Campos de hora	Valores
Minutos (*minute*)	0-59
Hora (*hour*)	0-23
Día del mes (*day or month*)	1-31
Mes (*month*)	1-12
Día de la semana (*day of week*)	0-7 (0 y 7 =Domingo)

Se siguen las siguientes reglas para la utilización de caracteres especiales en los campos de hora en el archivo Crontab:

- Utilizar un espacio para separar cada campo.
- Utilizar una coma para separar varios valores.
- Utilizar un guion para designar un rango de valores.
- Utilizar un asterisco como comodín para incluir todos los valores posibles.
- Utilizar una marca de comentario (#) al principio de una línea para indicar un comentario o una línea en blanco.

Estos campos presentan una serie de combinaciones que son descritas a continuación:

- **Minutos:** son los minutos en los cuales se va a ejecutar la tarea, estos tienen un rango de 0-59, son números enteros, presenta varias combinaciones como son:
  - **3-56** significa que la tarea se ejecutará desde el minuto 3 hasta el minuto 56.
  - ***** significa que la tarea se estará ejecutando todos los minutos, es decir que este atributo toma todos los valores posibles.
  - **5/5** significa que la tarea empezará a ejecutarse en el minuto 5 y se seguirá ejecutando cada 5 minutos.
  - **3,7,8** significa que la tarea empezará a ejecutarse en el minuto 3 y se ejecutará nuevamente en los minutos 7 y 8.
  - Estas combinaciones se pueden unir entre ellas.
- **Hora:** es la hora en que se ejecutará la tarea, esta tiene un formato de 24 horas, son números enteros y tiene un rango de 0-23, presenta varias combinaciones como son:
  - **02-04** significa que la tarea se ejecutará desde las 2:00 am hasta las 4:00am.
  - ***** significa que la tarea se estará ejecutando a todas horas, es decir que este atributo toma todos los valores posibles.

- **03/02** significa que la tarea empezará a ejecutarse a las 3:00am y se seguirá ejecutando cada 2 horas.
- **09,17,20** significa que la tarea empezará a ejecutarse a las 9:00am y se ejecutará nuevamente a las 5:00pm y a las 8:00pm.
- Estas combinaciones se pueden unir entre ellas.

- **Día del mes:** es el día del mes en el cual se va a ejecutar la tarea, presenta un rango de 1-31, son números enteros y presenta varias combinaciones como son:
  - **1-10** significa que la tarea se ejecutará desde el primer día del mes hasta el día 10.
  - ***** significa que la tarea se estará ejecutando todos los días del mes, es decir, este atributo toma todos los valores posibles.
  - **3/5** significa que la tarea empezará a ejecutarse el día 3 del mes y se seguirá ejecutando cada 5 días.
  - **4,8,12** significa que la tarea empezará a ejecutarse el día 4 y se ejecutará nuevamente los días 8 y 12.
  - Estas combinaciones se pueden unir entre ellas.
- **Mes:** es el mes en el cual se va a ejecutar la tarea, presenta un rango de 1-12, son números enteros y presenta varias combinaciones como son:
  - **2-4** significa que la tarea se ejecutará desde el primer día del mes de febrero hasta el mes de mayo.
  - ***** significa que la tarea se estará ejecutando todos los meses, es decir, este atributo toma todos los valores posibles.
  - **6/2** significa que la tarea empezará a ejecutarse en el mes de junio y luego se ejecutará cada dos meses.
  - **2,4,7** significa que la tarea empezará a ejecutarse el mes de febrero y se ejecutará nuevamente en los meses de abril y julio.
  - Estas combinaciones se pueden unir entre ellas.
- **Día de la semana:** es el día de la semana en el cual se va a ejecutar la tarea, presenta un rango de 0-7 siendo 0 y 7 valores correspondientes al domingo, son números enteros y presenta varias combinaciones como son:
  - **3-5** significa que la tarea se ejecutará desde el miércoles hasta el viernes.
  - ***** significa que la tarea se estará ejecutando todos los días de la semana, es decir, este atributo toma todos los valores posibles.
  - **1/2** significa que la tarea empezará a ejecutarse el lunes y luego se ejecutará cada dos días.

- **0,2,4,6** significa que la tarea empezará a ejecutarse el domingo y se ejecutará nuevamente los días martes, jueves y sábado.
- Estas combinaciones se pueden unir entre ellas.

A continuación de este formato se coloca el usuario que ejecutará el comando y posteriormente la(s) tarea(s) programadas que se desean realizar. Todos los comandos de un archivo Crontab se deben localizar en una sola línea, aunque esta sea demasiado larga. Cron reconoce algunas abreviaciones que reemplaza el formato de una tarea de Crontab. Corresponden a las opciones de programación más comunes:

- @yearly: una vez por año (1 de enero a las 00:00);
- @monthly: una vez por mes (el 1ro de cada mes a las 00:00);
- @weekly: una vez por semana (Domingo a las 00:00);
- @daily: una vez por día (a las 00:00);
- @hourly: una vez por hora (al principio de cada hora) (10).

1.2.2. Creación, visualización, edición y eliminación del archivo Crontab

Para la creación de un archivo Crontab la forma más sencilla es utilizando el comando crontab –e. Este comando hace un llamado al editor de texto que se encuentra predeterminado para el entorno del sistema en el cual se está trabajando. Al ser creado un archivo crontab este se almacenará por defecto en el directorio /var/spool/cron/crontabs y recibirá el nombre del usuario que lo creó. Se puede crear, visualizar, modificar, eliminar un archivo Crontab para otro usuario, o para *root*, si se tienen privilegios de superusuario, lo que significa que el usuario tiene todos los permisos. Luego de ser creado el archivo se añaden las líneas de comando descritas en la Tabla 2 acompañadas por el usuario y las tareas programadas. Después se verifican los cambios en el archivo con el comando crontab –l [nombre_de_usuario].

Para la creación de un archivo Crontab para otro usuario se utiliza el comando Crontab –e [nombre_de_usuario], donde nombre_de_usuario sería el usuario al cual se le está creando el archivo. Para verificar que para un usuario existe un archivo crontab, se utiliza el comando ls –l en el directorio /var/spool/cron/crontabs. Cuando se elimina un archivo crontab se emplea el comando crontab –r, a este comando se le especifica el archivo crontab que se quiere eliminar, de lo contrario eliminará el archivo crontab al usuario que se encuentre trabajando en él.

1.2.3. Control del acceso al comando Crontab

Se puede controlar el acceso al comando crontab, se realiza mediante dos archivos en el directorio /etc/cron.d: cron.deny y cron.allow. Estos archivos permiten que solo los usuarios especificados realicen tareas de comando crontab, como crear, editar, visualizar o eliminar sus propios archivos crontab. Los archivos cron.deny y cron.allow constan de una lista de nombres de usuario (un nombre de usuario por línea).

Los archivos de control de acceso funcionan de manera conjunta como se indica a continuación:

- Si cron.allow existe, solo los usuarios indicados en este archivo pueden crear, editar, visualizar o eliminar archivos crontab.
- Si cron.allow no existe, todos los usuarios pueden ejecutar archivos crontab, excepto los usuarios indicados en cron.deny.
- Si cron.allow y cron.deny no existen, se necesitan privilegios de superusuario para ejecutar el comando crontab.

Los privilegios de superusuario son necesarios para editar o crear los archivos cron.deny y cron.allow (12). Al concluir el estudio del servicio que permite la planificación de tareas se hace necesario el estudio de la herramienta a la cual va a ser integrado el módulo que se propone desarrollar.

1.3. Herramienta de Migración y Administración de Servicios Telemáticos

HMAST es una aplicación web que permite administrar los servidores de forma remota y local, contemplando las funcionalidades necesarias para administrar los usuarios y los servicios. Actualmente cuenta con los módulos de MySQL[1], DHCP[2], Bacula, Apache2[3] y OpenSSH[4] (13) (14). Para el desarrollo de esta aplicación se utilizó la metodología AUP-UCI, así como herramientas, tecnologías y lenguaje, como son Java, Spring, IntelliJ IDEA, Visual Paradigm, UML, Maven, Augeas y Git. A continuación, se describen las características principales de esta herramienta.

Funcionalidades que ofrece HMAST:

[1] MySQL sistema de gestión de base de datos relacional de código abierto, basado en lenguaje de consulta estructurado (SQL).
[2] DHCP protocolo de configuración dinámica de host, en inglés de *Dynamic Host Configuration Protocol*.
[3] Apache2 servidor web HTTP de código abierto.
[4] OpenSSH (*Secure SHell*, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red.

- Gestión de servidores lógicos: Permite la adición, edición, eliminación de los datos de un servidor lógico, además permite la conexión remota y desconexión a un servidor seleccionado.
- Gestión de servicios telemáticos asociados a un servidor lógico: Permite la adición, edición, eliminación de los datos de un módulo, así como activación y desactivación de los mismos.
- Gestión de las variables de configuración asociadas a un servidor lógico: Permite cargar y salvar las variables de configuración de los servicios telemáticos encontrados en un servidor lógico (ficheros de configuración, nombre de módulos, demonios, entre otros) (15).

1.3.1. Arquitectura de HMAST

La arquitectura que utiliza esta herramienta es N-Capas orientada al Dominio (ver Figura 1). Su estructura se basa en cinco capas, las cuales interactúan a través de interfaces utilizando la inyección de dependencias. Las características y principales responsabilidades de cada una de estas capas se describen a continuación.

• **Capa Presentación (*Presentation*)**: Presenta al usuario los conceptos del negocio mediante una interfaz de usuario. Además, facilita la explotación de dichos procesos, informa sobre la situación de los procesos de negocio e implementación de las reglas de validación de dicha interfaz. En esta capa se realiza el tratamiento a las excepciones lanzadas desde capas inferiores.

• **Capa Aplicación (A*pplication*)**: Realiza las llamadas a servicios de la capa inferior y tiene la responsabilidad de adaptar la información que recibe a los requisitos de los servicios de dominio. En esta capa también se ubican operaciones de trazas, seguridad, envío de correos electrónicos, cuando no forman parte estricta del negocio. Está compuesta por los servicios de aplicación y los objetos para la transferencia de datos (DTO, del inglés *Data Object Transfer*).

• **Capa Dominio (*Domain*):** Se encarga de implementar la lógica de dominio, o sea, las reglas del negocio. Define las interfaces de persistencia a datos, conocidos como contratos a repositorios, pero no los implementa. Es responsable de realizar las validaciones y sus componentes solo dependen de la capa Infraestructura Transversal. Esta capa está compuesta por las entidades del dominio, los servicios de dominio y los contratos de repositorio. Las entidades representan objetos del dominio y están definidas fundamentalmente por su identidad y continuidad en el tiempo. Los servicios de dominio contienen la lógica que trata a las entidades como un todo y los contratos de repositorios son interfaces que especifican las operaciones que deben implementar los repositorios.

• **Capa Persistencia (*Persistence*):** Es responsable de contener el código necesario para persistir los datos. Se compone fundamentalmente por los repositorios, que son clases que implementan los contratos de repositorios definidos en la capa de Dominio.

• **Capa Infraestructura Transversal (*Infrastructure Crosscutting*)**: Es responsable de promover la reutilización de código. Contiene las operaciones de seguridad, inicios de sesión, monitoreo del sistema, mecanismos de persistencia reutilizables, validadores genéricos y todas aquellas operaciones que se puedan utilizar desde otras capas (16).

Figura 1. Arquitectura de HMAST.

1.3.2. Consideraciones para implementar un módulo para HMAST

Para la implementación de un módulo para HMAST se debe tener en consideración algunos aspectos como los que se describen a continuación:

- La lógica de Aplicación no deberá incluir ninguna lógica del Dominio, solo tareas de coordinación relativas a requerimientos técnicos de la aplicación, como conversiones de formatos de datos de entrada a entidades del Dominio y llamadas a componentes de Infraestructura para que realicen tareas complementarias.
- La capa de Presentación no debe tener como entrada o salida, objetos de dominio, sino objetos para la transferencia de datos (DTO).
- Las entidades solo pueden tener dependencias de componentes de la capa de Dominio.
- Las clases de servicios deben ser las únicas responsables de acceder a los repositorios, no se debe implementar código de persistencia a datos en la capa de Dominio.
- Solo se puede acceder a la información almacenada en los servidores haciendo uso de los repositorios.

- El código reutilizable por más de un repositorio debe estar disponible en la capa de Infraestructura Transversal (16).

Se hace necesario del estudio de sistemas que implementen la planificación de tareas para conocer si los mismos pueden ser integrados con la herramienta.

1.4. Sistemas que implementan la planificación de tareas

En el presente epígrafe se realiza un estudio de tres sistemas de código abierto que permiten la planificación de tareas.

1.4.1. Gnome Schedule

Gnome Schedule (Ver Figura 2) es una herramienta que permite programar tareas mediante la utilización de archivos Crontab y una interfaz gráfica. Esta herramienta no se encuentra instalada de forma predeterminada, pero sí se incluye en los repositorios. Está escrita en *Python* y se puede encontrar en varias distribuciones de Linux.

Entre sus principales características se encuentran:

- Permite añadir títulos e íconos a las tareas programadas.
- Permite crear plantillas, de manera que no se tienen que introducir los datos sobre cuándo ejecutarse una y otra vez.
- Al ejecutarse como *root*, permite editar el Crontab y las tareas "at" de cualquier usuario.
- Cadenas legibles para saber cuándo ocurren los eventos planificados.
- Tiene un modo avanzado para los expertos en Crontab.
- Contiene expresiones comunes predefinidas del tipo: cada minuto, todas las semanas, mañana, la próxima semana.
- Posee un calendario para ayudar a elegir el día en que se quiere ejecutar la tarea (17).

Figura 2. Ícono de Gnome Shedule.

1.4.2. Webmin

Webmin es una herramienta de configuración de sistemas Linux accesible vía web con una interfaz gráfica. Fue desarrollada por Jamie Cameron, aunque muchas personas han contribuido con parches y traducciones a otros idiomas. Utiliza cualquier navegador que admita tablas, formularios y Java para el módulo Administrador de archivos. Todas las recientes versiones de Webmin se encuentran bajo la licencia BSD[5], lo que implica que pueden ser libremente distribuidas y modificadas para uso comercial y no comercial.

Se caracteriza por ser modular, lo que significa que es posible añadir nuevas funcionalidades fácilmente. Brinda la posibilidad de configurar aspectos internos de muchos sistemas operativos, como usuarios, cuotas de espacio, servicios, archivos de configuración, apagado del equipo, así como modificar y controlar muchas aplicaciones *open source*.

Webmin consiste en un servidor web, y una serie de programas con interfaz de entrada común CGI (*Common Gateway Interface*), se actualizan los archivos directamente al sistema como /etc/inetd.conf y /etc/passwd. El servidor web y todos los programas CGI están escritos en Perl versión 5, y no usan módulos de Perl que no sean estándar. La última versión disponible es la 1.820. A través de esta herramienta se pueden manejar muchos aspectos del sistema remotamente, además de programas como son Apache, MySQL, PHP[6], DHCP, Samba[7], ProFTPD[8] (18) (19). Entre los módulos de Webmin se encuentra Tareas Planificadas de Cron *(Scheduled Cron Jobs*), ubicado en la categoría de sistema, es utilizado para programar tareas de Cron desde la interfaz gráfica. Se puede activar, desactivar, crear nuevas tareas Cron, y otras acciones desde Webmin (20).

1.4.3. Nova NAS

Nova NAS es una aplicación web para la administración de un servidor de almacenamiento en la red (NAS), desplegada con Nova Servidores 2015, que gestiona los servicios y herramientas de forma centralizada y provee una interfaz intuitiva. Esta aplicación web provee una interfaz para la administración de tareas programadas.

Algunas de sus principales características son:

---

[5] BSD: *Berkeley Software Distribution* licencia que permite ver el código y modificarlo, pero también permite cerrar el sistema o la aplicación.

[6] PHP son las siglas en inglés de “Hypertext Pre-Processor” que al traducirlo al español significa “Lenguaje de Programación Interpretado”.

[7] Samba herramienta de *software* libre y código abierto.

[8] ProFTPD Servidor FTP.

- Desplegada sobre Nova Servidores 2015.
- Administración basada en la web.
- Soporte multilenguaje.
- Licencia Pública General de GNU o más conocida por su nombre en inglés *General Public License.*
- Gestión de tareas programadas, Nova NAS provee una interfaz para la administración de tareas programadas.
- Soporte para conexión segura.
- Administración remota (SSH, Secure SHell).
- Transferencia de archivos (FTP, Protocolo de Transferencia de Archivos por sus siglas en inglés File Transfer Protocol).
- Compartición de Windows (SMB, Server Message Block).
- Compartición en la red (NFS, Sistema de archivos de red por sus siglas en inglés Network File System).
- Copias sincronizadas (Rsync, Remote synchronize) (21).

1.4.4. Resultado del estudio de sistemas similares existentes

Luego de realizar un estudio a los sistemas que realizan la planificación de tareas se arriba a las siguientes conclusiones:

- Gnome Shedule es una aplicación de escritorio, lo que imposibilita la integración de este con HMAST.
- Webmin administra los servicios de manera local, es decir para ser administrados esta herramienta tiene que estar previamente instalada en el *host* que vaya a administrar por lo que es muy costoso en tiempo.
- Webmin no detecta cuando hay cambios en los ficheros de configuración de forma manual, afectado el servicio o eliminado el archivo que no reconozca.
- Nova NAS no abarca todos los elementos del servicio Cron, pero una forma de representación de la arquitectura de la información que se tendrá en cuenta para la implementación del módulo.

Luego del estudio realizado se propone desarrollar un módulo para la herramienta HMAST que permita la planificación de tareas debido a que las aplicaciones y herramientas estudiadas no pueden ser integradas a HMAST ya que no cumplen con lo expuesto en el epígrafe 1.3.2.

1.5. Metodología, lenguaje, herramientas y tecnologías

El módulo a desarrollar será integrado a la herramienta HMAST por lo que la metodología, lenguaje, herramientas y tecnologías se corresponderán a las que usa la herramienta principal y se describen a continuación.

1.5.1. Metodología de desarrollo de *software*

Las metodologías de desarrollo de *software* son el conjunto de procedimientos, técnicas, herramientas y un soporte documental, que ayuda a los desarrolladores a realizar nuevo *software*. La metodología define quién debe hacer qué, cuándo y cómo debe hacerlo para obtener los distintos productos parciales y finales (22).

Para el desarrollo del módulo de planificación de tareas se emplea la metodología AUP-UCI, resultante de una variación de la metodología Proceso Unificado Ágil (AUP). AUP-UCI define 3 fases, 7 disciplinas y 11 roles. Las fases de AUP-UCI son inicio, ejecución y cierre. Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación. En la segunda fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. En la fase de cierre se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

Los roles definidos por esta metodología son: jefe de proyecto, planificador, analista, arquitecto de información (opcional), desarrollador, administrador de la configuración, cliente/proveedor de requisitos, administrador de calidad, probador, arquitecto de sistema y administrador de base de datos. Además, AUP-UCI define cuatro escenarios para modelar el sistema. Se emplea el escenario número 4, el cual indica que proyectos que no modelen negocio solo pueden modelar el sistema con Historias de Usuarios (HU) (23).

1.5.2. Lenguaje de programación

Un lenguaje de programación es una técnica estándar de comunicación diseñada para expresar procesos que puedan ser ejecutados en una computadora. Está conformado por un conjunto de reglas sintácticas y semánticas que definen un programa informático. Un lenguaje de programación permite a un programador especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados y transmitidos, además de qué acciones debe tomar bajo una variada gama de circunstancias (24).

Java fue diseñado como un lenguaje orientado a objetos, los cuales agrupan en estructuras encapsuladas tanto sus datos como las funcionalidades. Proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets[9] y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas. Es interpretado y compilado a la vez. Proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores. Soporta sincronización de múltiples hilos de ejecución a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas (25).

1.5.3. Framework

Un *framework* (marco de trabajo) es un conjunto de bibliotecas, módulos o artefactos reutilizables que establecen una estructura que brinda soporte a los desarrolladores. Provee las piezas, las cuales deben ser personalizadas y conectadas para desarrollar la aplicación. Representa un conjunto de consideraciones especiales que garantizan la calidad del diseño del mismo (26).

Spring es un *framework* de desarrollo de código abierto. Fue creado para hacer frente a la complejidad del desarrollo de aplicaciones empresariales. Es una plataforma Java que proporciona un amplio soporte de infraestructura para el desarrollo de aplicaciones. Sus características principales son la inyección de dependencias, que tiene como objetivo lograr un bajo acoplamiento entre los objetos de la aplicación y la programación orientada a aspectos, la cual se trata de un paradigma de programación que intenta separar las funcionalidades secundarias de la lógica del negocio (27).

1.5.4. Entorno de Desarrollo Integrado

Un Entorno de Desarrollo Integrado (IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDE proporcionan un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java, C#, Delphi, Visual Basic. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto (28).

---

[9] Los sockets constituyen el mecanismo para la entrega de paquetes de datos provenientes de la tarjeta de red a los procesos o hilos apropiados. Un socket queda definido por un par de direcciones IP local y remota, un protocolo de transporte y un par de números de puerto local y remoto.

IntelliJ IDEA es un IDE para el lenguaje de programación Java diseñado específicamente para maximizar la productividad del desarrollador. Es multiplataforma, presenta un mayor soporte a lenguajes, entre los que destacan:

- PHP, Python y Ruby.
- SQL, incluyendo PostgreSQL, MySQL, Oracle, SQL Server.
- Facilita el desarrollo con Spring MVC, Webflow, Jugar, Grails, Servicios Web, JSF, Struts, Flex.
- Incluye asistencia de código final para HTML5, CSS3, SASS, MENOS, JavaScript, CoffeScript, Node.js, ActionScript y otros lenguajes (29).

1.5.5. Herramienta de Ingeniería de *Software* Asistida por Computadora

La Ingeniería de S*oftware* Asistida por Computadora (o CASE) es un conjunto de herramientas de programación que utilizan una interfaz común para diseñar, desarrollar y depurar *software*. Por lo tanto, un entorno CASE consta de herramientas que proveen un modelo visual de una aplicación, herramientas que crean un código a través de interfaces visuales y finalmente un depurador para probar el código final (30).

Visual Paradigm para el Lenguaje Unificado de Modelado (UML), es una herramienta para desarrollo de aplicaciones utilizando modelado UML, utilizada por ingenieros de *software*, analistas de sistemas y arquitectos de sistemas que son encargados de la construcción de sistemas a gran escala y tiene la necesidad de confiabilidad y estabilidad en el desarrollo orientado a objetos.

Visual Paradigm también ofrece:

- Navegación intuitiva entre la escritura del código y su visualización.
- Generador de informes en formato PDF/HTML.
- Documentación automática *Ad-hoc*.
- Ambiente visualmente superior de modelado.
- Sofisticado diagramador automáticamente de *layout.*
- Sincronización de código fuente en tiempo real u *on deman* (31).

1.5.6. Lenguaje Unificado de Modelado

UML es uno de los estándares *Object Management Group* (OMG) o “Grupo de Administración de Objetos” más usados y se ha adoptado a nivel internacional por numerosos organismos (32). Diseñado para visualizar, especificar, construir y documentar *software* orientado a objetos (33). UML está compuesto por diversos

elementos gráficos que se combinan para conformar diagramas, por lo que cuenta con reglas para combinar tales elementos (34).

1.5.7. Herramienta para la gestión y construcción de proyectos

Maven es una herramienta de *software* de código abierto para la gestión y construcción de proyectos basada en estándares. Permite gestionar el ciclo de vida de un proyecto desde su creación hasta la generación de un binario que pueda distribuirse con este. Ofrece facilidades a los desarrolladores como la sencilla y ágil creación de proyectos o módulos, además de aplicar una estandarización de la estructura y organización del proyecto. También dispone de un mecanismo de gestión de dependencias de un proyecto sobre las bibliotecas propias o de terceros y mantiene disponible para los desarrolladores un repositorio de bibliotecas de código abierto en constante actualización (16).

1.5.8. Herramienta para administrar ficheros de configuración

Augeas es una herramienta de edición de archivos configuración. Esta herramienta analiza archivos de configuración en sus formatos nativos y los transforma en un árbol. Los cambios de configuración se realizan manipulando este árbol y guardándolo en archivos de configuración nativos. Augeas es una herramienta modular, pero a diferencia de otras herramientas modulares esta depende totalmente de sus módulos, debido a que son los encargados de permitir la transformación de un fichero en árbol y viceversa. A estos módulos se le llaman Lenses y son los encargados de brindar soporte a cada fichero de configuración, pues cada uno describe un archivo (35).

1.5.9. Sistema de control de versiones

Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en uno o más proyectos, almacenando así versiones del producto en todas sus fases del desarrollo. Las versiones son como fotografías que registran su estado en ese momento del tiempo y se van guardando a medida que se hacen modificaciones al código fuente. Se utiliza Git como sistema de control de versiones, esta herramienta lanzada en el año 2005, es rápida y eficiente con grandes proyectos. Git modela sus datos como un conjunto de instantáneas de un mini sistema de archivos (36).

Conclusiones parciales

El estudio de las características, funcionalidades, procedimientos de Cron y Crontab posibilitó un mayor conocimiento de la planificación de tareas. Se describe HMAST, lo que permitió conocer cómo realiza su funcionamiento, así como los aspectos a tener en cuenta en el diseño y la implementación de la solución. Se estudiaron varios sistemas que permiten la planificación de tareas, concluyendo que para el diseño del módulo se utilizarán algunas características de Nova NAS y el resto de los sistemas no podrán ser utilizados.

Capítulo 2: Diseño del módulo de planificación de tareas para HMAST

En el presente capítulo se describe la propuesta del módulo en cuestión, tomando como punto inicial el sistema principal HMAST y los elementos planteados por la metodología AUP-UCI. Se dan a conocer las principales funcionalidades de la propuesta de solución mediante requisitos funcionales descritos a través de Historias de Usuario. Se muestran prototipos de interfaz de usuario, así como los patrones de diseño que se utilizarán durante la implementación.

2.1. Propuesta de solución

Para darle solución al problema planteado se propone desarrollar un módulo para HMAST que administre el servicio Cron. El módulo a desarrollar permitirá, mediante una interfaz gráfica, la planificación de tareas personalizadas, visualizar y editar la ejecución de las tareas que se ejecutan a cada hora, diariamente, semanalmente y mensualmente, integrará también la gestión de las ejecuciones y grupos de tareas. Todo esto será posible debido a que la aplicación realizará conexiones seguras mediante el uso del protocolo SSH al servidor, accederá a los ficheros de configuración y los modificará de acuerdo a las especificaciones del usuario.

2.2. Artefactos generados

El desarrollo de la propuesta de solución se encuentra guiado por la metodología AUP-UCI. No se realiza el modelado del negocio y las funcionalidades se describen en un documento de Especificación de Requisitos de *Software*, todo esto corresponde al escenario 4 que establece esta metodología. Por lo que el principal artefacto generado en el proceso de desarrollo lo constituyen las Historias de Usuario en las cuales los requisitos son encapsulados.

2.2.1. Especificación de requisitos

Los requisitos son características requeridas por el sistema, posibilitan cumplir con un objetivo o solucionar un problema determinado. Los requisitos son necesarios para el cumplimiento de las especificaciones del cliente (37).

2.2.1.1. Requisitos funcionales

Los requisitos funcionales de un sistema describen lo que el sistema debe hacer. Estos requerimientos dependen del tipo de *software* que se desarrolle, de los posibles usuarios del *software* y del enfoque general tomado por la organización al redactar requerimientos (38). La Tabla 3 muestra nombre, descripción y prioridad de los requisitos funcionales del módulo a desarrollar.

Tabla 3. Listado de requisitos funcionales.

No.	Nombre	Descripción	Prioridad
RF1	Iniciar el servicio Cron	Permite iniciar el servicio Cron el cual ya se encuentra instalado automáticamente en el servidor.	Alta
RF2	Reiniciar el servicio Cron	Permite reiniciar el servicio Cron.	Alta
RF3	Detener el servicio Cron	Permite detener el servicio Cron.	Alta
RF4	Aplicar cambios al servidor	Permite aplicar los cambios que se realizan al servidor.	Alta
RF5	Descartar cambios realizados localmente	Permite descartar los cambios que fueron realizados localmente.	Alta
RF6	Crear tarea personalizada	Permite crear una tarea personalizada.	Alta
RF7	Listar tareas personalizadas	Permite listar las tareas personalizadas que hayan sido creadas hasta el momento.	Alta
RF8	Editar tarea personalizada	Permite editar los campos que conforman la tarea personalizada previamente seleccionada.	Alta
RF9	Eliminar tareas personalizadas	Permite eliminar las tareas personalizadas que hayan sido previamente seleccionadas.	Alta
RF10	Listar ejecuciones realizadas a cada hora	Permite listar los *scripts* que serán ejecutados a cada hora	Media
RF11	Adicionar ejecución realizada a cada hora	Permite adicionar un *scripts* que será ejecutado a cada hora	Media
RF12	Eliminar ejecuciones realizadas a cada hora	Permite eliminar los *scripts* que hayan sido previamente seleccionados.	Media
RF13	Editar tiempo en que se realizan las ejecuciones por hora	Permite editar el tiempo en que se realizan las ejecuciones por hora.	Media

RF14	Listar ejecuciones realizadas diariamente	Permite listar los *scripts* que serán ejecutados diariamente.	Media
RF15	Adicionar ejecución realizada diariamente	Permite adicionar un *scripts* que será ejecutado diariamente.	Media
RF16	Eliminar ejecuciones realizadas diariamente	Permite eliminar los *scripts* que hayan sido previamente seleccionados.	Media
RF17	Editar tiempo en que se realizan las ejecuciones diarias	Permite editar el tiempo en que se realizan las ejecuciones diarias.	Media
RF18	Listar ejecuciones realizadas semanalmente	Permite listar los *scripts* que serán ejecutados semanalmente.	Media
RF19	Adicionar ejecuciones realizadas semanalmente	Permite adicionar un *scripts* que será ejecutado semanalmente.	Media
RF20	Eliminar ejecuciones realizadas semanalmente	Permite eliminar los *scripts* que hayan sido previamente seleccionados.	Media
RF21	Editar tiempo en que se realizan las ejecuciones semanales	Permite editar el tiempo en que se realizan las ejecuciones semanales.	Media
RF22	Listar ejecuciones realizadas mensualmente	Permite listar los *scripts* que serán ejecutados mensualmente.	Media
RF23	Adicionar ejecuciones realizadas mensualmente	Permite adicionar un *scripts* que será ejecutado mensualmente.	Media
RF24	Eliminar ejecuciones realizadas mensualmente	Permite eliminar los *scripts* que hayan sido previamente seleccionados.	Media
RF25	Editar tiempo en que se realizan las ejecuciones mensuales	Permite editar el tiempo en que se realizan las ejecuciones mensuales.	Media
RF26	Crear grupo de tareas	Permite crear un grupo de tareas	Baja
RF27	Listar grupo de tareas	Permite listar un grupo de tareas	Baja
RF28	Editar grupo de tareas	Permite editar un grupo de tareas	Baja
RF29	Eliminar grupo de tareas	Permite eliminar un grupo de tareas	Baja

2.2.1.2. Requisitos no funcionales

Los requisitos no funcionales restringen el sistema en desarrollo y el proceso de desarrollo que se debe utilizar. Pueden ser requerimientos del producto, organizacionales o externos. A menudo están relacionados con las

propiedades emergentes del sistema y, por lo tanto, se aplican al sistema completo (38). Los requisitos no funcionales son heredados de la herramienta HMAST, debido a que estos condicionan el funcionamiento del módulo para lograr una correcta integración con el sistema base. La Tabla 4 describe los requisitos no funcionales del módulo a desarrollar.

Tabla 4. Listado de requisitos no funcionales.

No.	Nombre	Descripción	Atributo de Calidad
RnF1	Proteger información y los datos, para que personas o sistemas desautorizados no puedan leer o modificar los mismos, y las personas o sistemas autorizados tengan el acceso a ellos.	Todos los datos manejados por el módulo estarán encriptados; tanto los que se transfieren entre la estación cliente y el servidor HMAST como los que se almacenan temporalmente en el servidor.	Seguridad (Acceso restringido)
RnF2	Permitir al usuario aprender su aplicación	Internacionalizar la información que se muestra, en los idiomas español e inglés.	Usabilidad
RnF3	Emplear como lenguaje de programación Java.	Estas son restricciones heredadas de HMAST, el módulo debe cumplirlas para poder integrarse correctamente.	Funcionabilidad
RnF4	El módulo se ejecutará sobre el sistema operativo GNU/Linux Nova Servidores.		
RnF5	Disponer en el sistema operativo GNU/Linux de los siguientes paquetes: augeas-tools, libjna-java, openjdk-7-jdk.		
RnF6	El módulo debe mantener un nivel de ejecución o desempeño especificado en caso de fallos del *software* o de infracción de su interfaz especificada.	Ante el fallo de una funcionalidad del sistema el resto de las funcionalidades que no dependen de esta, deberán seguir funcionando.	Confiabilidad (Tolerancia a fallos)

2.2.2. Historias de Usuario

Las Historias de Usuario son el instrumento principal para identificar requerimientos de usuario. Son descripciones cortas y simples de una funcionalidad, escritas desde la perspectiva de la persona que necesita

una nueva capacidad de un sistema, por lo general el usuario, área de negocio o cliente (39). A continuación, se describen las HU para los requisitos funcionales siguientes:

- Iniciar el servicio Cron.
- Crear tarea personalizada.

El resto de las HU se encuentran en el Anexo 1.

Tabla 5. Historia de Usuario iniciar el servicio Cron.

Número: 1	**Nombre del requisito:** Iniciar el servicio Cron.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 horas
Descripción: Permite iniciar el servicio Cron. La funcionalidad inicia cuando el usuario selecciona el botón iniciar el servicio. Para esto el sistema ejecutará el comando *$ sudo service cron start*.	
Observaciones: El servicio debe de estar detenido.	
Prototipo elemental de interfaz gráfica de usuario: 	

Tabla 6. Historia de Usuario Crear tarea personalizada.

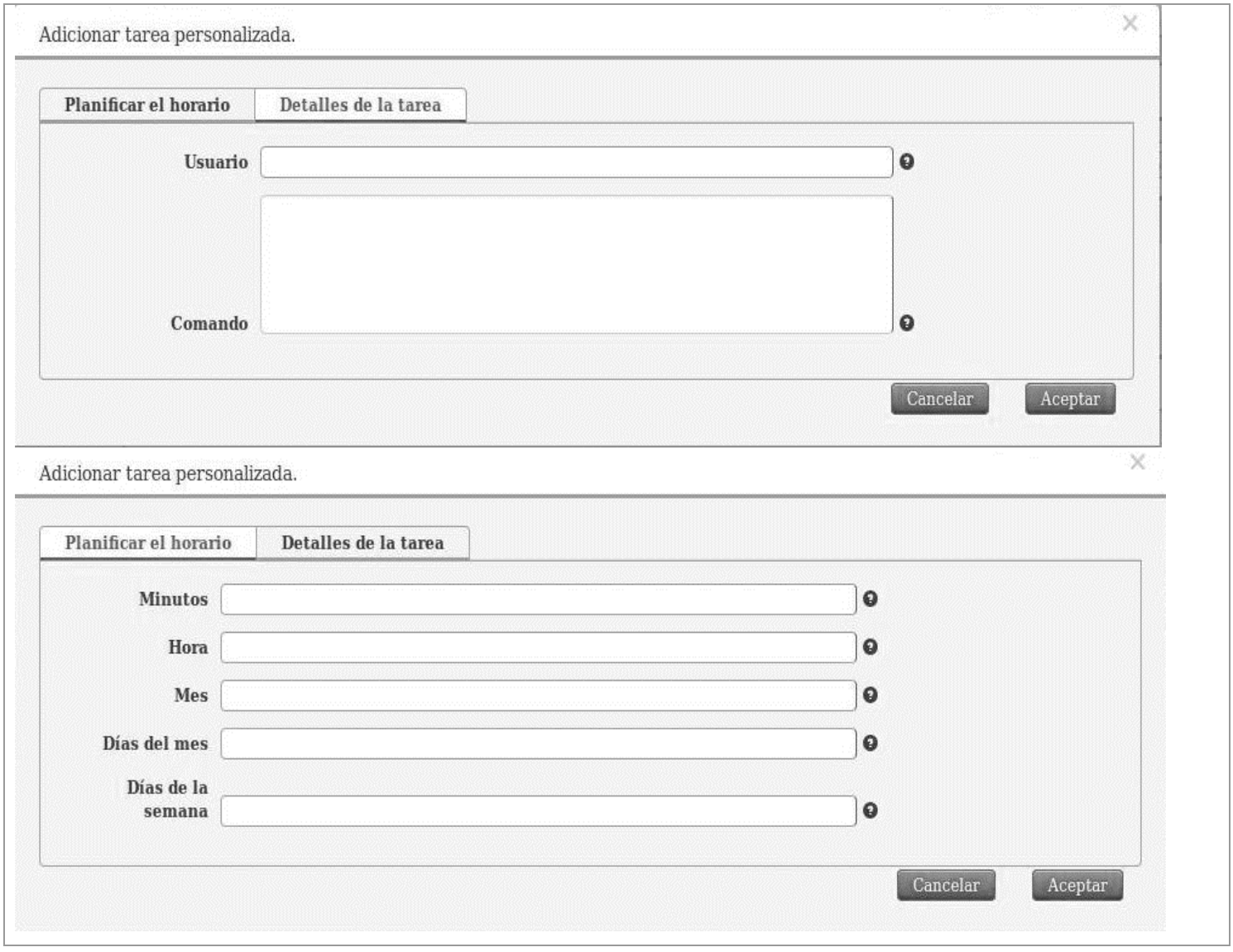
Número: 6	**Nombre del requisito:** Crear tarea personalizada.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 horas
Descripción: Permite crear una nueva tarea personalizada. La funcionalidad inicia cuando el usuario selecciona la opción *Tarea Personalizada*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página y luego selecciona el botón *Adicionar* que se encuentra en la parte superior derecha. Al instante se muestra una interfaz con un conjunto de atributos referentes a la nueva tarea que se desea crear. Los atributos son: • **Minutos:** son los minutos en los cuales se va a ejecutar la tarea. Permite caracteres como (*), (,), (/), (-) y números y tiene un rango de 0-59. Es un campo de texto.	

- **Hora:** es la hora en que se ejecutara la tarea, este tiene un formato de 24 horas, son números enteros y tiene un rango de 0-23. Permite caracteres como (*), (,), (/), (-). Es un campo de texto.
- **Día del mes:** es el día del mes en el cual se va a ejecutar la tarea, presenta un rango de 1-31, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto.
- **Mes:** es el mes en el cual se va a ejecutar la tarea, presenta un rango de 1-12, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto.
- **Día de la semana:** es el día de la semana en el cual se va a ejecutar la tarea, presenta un rango de 0-7 siendo 0 y 7 valores correspondientes a el domingo, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto.
- **Usuario:** es el usuario que va a ejecutar la tarea que se está creando. Es un campo de texto y es obligatorio. Permite hasta 255 caracteres de a-z.
- **Comando:** es el comando que va a ser ejecutado. Es un campo de texto y es obligatorio. Permite hasta 1037 caracteres.

Observaciones:

Esta nueva tarea se adiciona al final del archivo */etc/crontab.*

Prototipo elemental de interfaz gráfica de usuario:



2.3. Arquitectura del módulo

El diseño de la arquitectura de un sistema es el proceso por el cual se define una solución para los requisitos técnicos y operacionales del mismo. Este proceso define qué componentes formarán el sistema, cómo se relacionan entre ellos, y cómo mediante su interacción llevan a cabo la funcionalidad especificada, cumpliendo con los criterios de calidad indicados como seguridad y usabilidad (40).

La arquitectura del módulo es la definida para la herramienta HMAST para lograr una consistencia con los componentes del sistema (ver Figura 3). Es utilizada la arquitectura N-Capas orientada al Dominio, la cual

presenta como objetivo estructurar de forma clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura siguiendo el patrón N-Capas y las tendencias de arquitecturas orientadas al dominio (40). En las capas *Presentation, Application, Domain* y *Persistence* se incorpora el paquete Cron el cual contendrá todo lo referente al módulo de planificación de tareas.

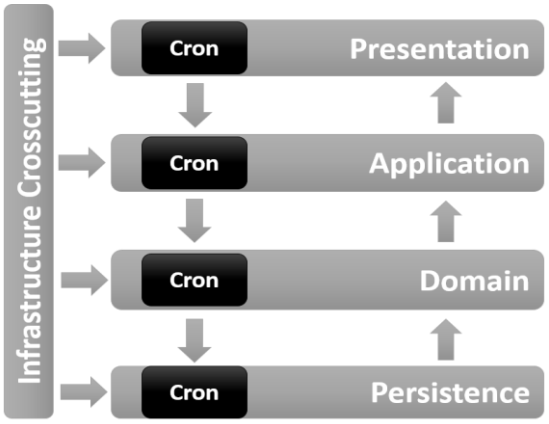


Figura 3. Arquitectura del módulo.

2.4. Diagrama de paquetes

Un diagrama de paquetes permite dividir un modelo para agrupar y encapsular sus elementos en unidades lógicas individuales. Cada paquete puede contener otros paquetes o clases, que tienen interfaces y realizan cierta funcionalidad (41). En el diseño del módulo se toma como referencia la arquitectura propuesta anteriormente, en la que a cada capa le corresponde un paquete y dentro de este se encuentra otro, nombrado Cron en el cual se encuentran todas las clases correspondientes al módulo (ver Figura 4).

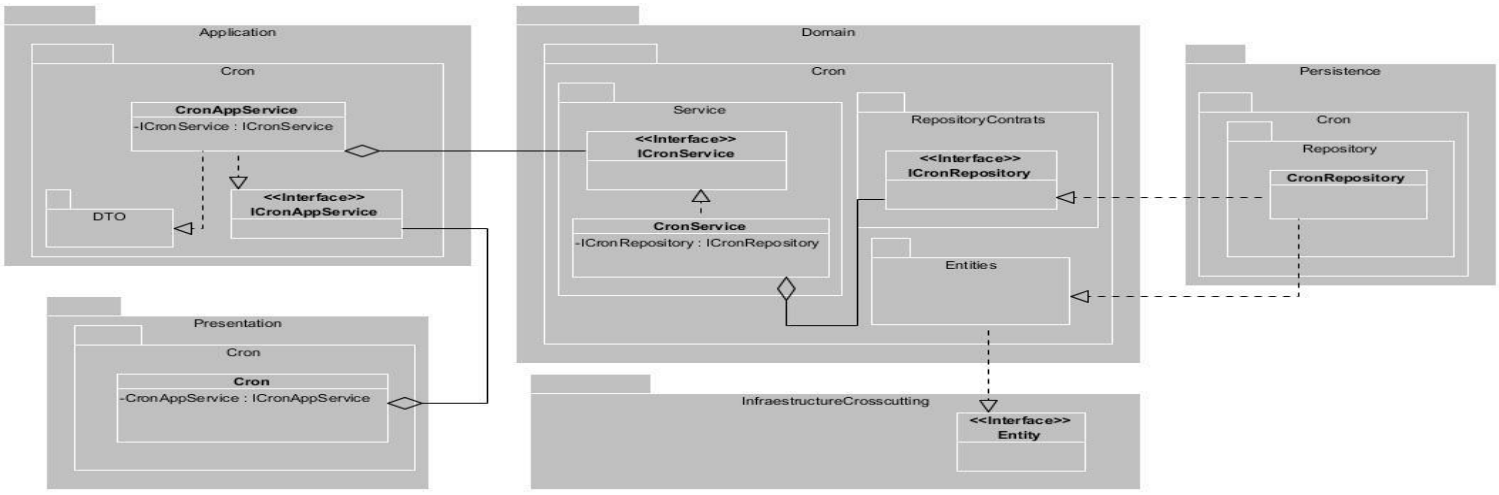


Figura 4. Diagrama de paquetes del módulo.

A continuación, se realiza una breve descripción del funcionamiento del módulo a través de paquetes que este presenta y la organización de sus elementos.

La capa de Presentación (*Presentation*) está compuesta por el paquete Cron y dentro de este se encuentra la clase controladora con, que a través de peticiones construye las interfaces de usuario (que se encuentran dentro de este paquete) y las comunica con la capa de Aplicación.

La capa de Aplicación (*Application*) está compuesta por el paquete Cron y dentro de este el paquete DTO. En Cron se encuentra la clase *interface ICronAppService*, en esta se encuentran las funcionalidades que son llamadas desde Presentación y serán implementados en la clase *CronAppService*. El paquete DTO se encarga de la transferencia de datos entre la capa de Aplicación y la capa de Dominio.

La siguiente capa es la de Dominio (*Domain*), en esta capa se localiza como subpaquete Cron, la que contiene a su vez tres paquetes nombrados *Entities, Services* y *RepositoryContrats.* En el subpaquete *Entities* se encuentran las clases entidades que son las contenedoras de toda la información referente al servicio Cron. El paquete *Services* contiene la clase *ICronService* que define funcionalidades que son accedidas desde la capa de Aplicación, implementadas por la clase *CronService*, la cual realiza las validaciones de los datos antes de realizar las operaciones en el Repositorio. También se encuentra el paquete R*epositoryContrats*, en el que se definen los contratos de repositorios en *ICronRepository,* pero no realiza su implementación.

La capa de Persistencia (*Persistence*) es la siguiente, en la que se halla el paquete Cron y dentro el paquete Repository en el que se encuentra la clase *CronRepository*, la cual implementa a la clase *ICronRepository* definida en el subpaquete *RepositoryContrats* en el paquete *Domain*. Esta capa trabaja directamente con los

ficheros de configuración del servicio Cron, contiene el código necesario para persistir los datos, lo que significa que los cambios se muestren al usuario en el propio momento en que se realizan.

2.5. Patrones de diseño

Los patrones de diseño son soluciones para problemas típicos y recurrentes que se pueden encontrar a la hora de desarrollar una aplicación. Aunque una aplicación sea única, tendrá partes comunes con otras aplicaciones: acceso a datos, creación de objetos, operaciones entre sistemas. Se pueden solucionar problemas utilizando algún patrón, ya que son soluciones probadas y documentadas por multitud de programadores (42). Para la implementación del módulo a desarrollar se hizo uso de los patrones GRASP[10] y GoF[11].

2.5.1. Patrones GRASP

Los patrones GRASP describen los principios fundamentales para asignar responsabilidades a los objetos, recomendados en el diseño de *software* (43). A continuación, se describen los patrones que fueron utilizados en el desarrollo del módulo propuesto.

- **Experto en información o experto:** se utiliza para la asignación de responsabilidades relacionadas con la obtención de información. Conduce a diseños donde los objetos del *software* realizan aquellas operaciones que normalmente se hacen a los objetos inanimados del mundo real que representan. Se mantiene el encapsulamiento de la información logrando un bajo acoplamiento entre los objetos y se distribuye el comportamiento entre las clases, lo que estimula las definiciones de clases más cohesivas. Este patrón se emplea en las distintas capas del módulo ya que cada una de ellas tiene una responsabilidad asignada la cual es encargada de implementarla.
- **Creador:** guía la asignación de responsabilidades relacionadas con la creación de objetos. La intención básica es encontrar un creador que necesite conectarse al objeto creado en alguna situación; eligiéndolo como el creador se favorece el bajo acoplamiento. Su utilización se evidencia cuando en la clase *CronAppService* de la capa de Aplicación se realiza la creación de los DTO utilizando para ello las entidades necesarias del dominio. Esta clase es la encargada de crear un DTO a partir de una entidad o crear la entidad a partir de un DTO.

---

[10] GRASP: Patrones Generales de *Software* para Asignar Responsabilidades, del inglés General Responsibility Assignment *Software* Patterns.
[11] GoF: Patrones de diseño de *software*, acrónimo de Gang of Four, Pandilla de los Cuatro, en su traducción al español.

- **Bajo acoplamiento:** consiste en asignar responsabilidades de manera que el acoplamiento permanezca bajo. El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. El uso de este patrón permite la reutilización de las clases y que no se afecten por cambios que se realicen en otros componentes. Este patrón se emplea en las distintas capas del módulo mediante el uso de interfaces que relacionan una capa con otra de forma que dichas relaciones no se establezcan directamente hacia las clases. Las conexiones se realizan a través del mecanismo de inyección de dependencias. Esto se evidencia en el paquete *RepositoryContrats*, donde se define la interfaz *ICronRepository*, y la implementación de sus métodos es realizada por la clase *CronRepository* en la capa de Persistencia.
- **Alta cohesión:** La cohesión es la medida en la que un componente se dedica a realizar solo la tarea para la cual fue creado, delegando las tareas complementarias a otros componentes. (Una clase debe de hacer lo que respecta a su entidad, y no hacer acciones que involucren a otra clase o entidad) (44).Este patrón se evidencia en la clase *CronRepository* debido a que implementa métodos específicos como el *loadServiceConfiguation* para leer y escribir en los ficheros de configuración que se ejecuten.

2.5.2. Patrones GoF

Los patrones GoF son patrones de diseño de *software* que solucionan problemas de creación de instancias, estos contribuyen a encapsular y abstraer dicha creación (45).

- **Patrón Solitario (*Singleton*):** Garantiza la existencia de una única instancia para una clase. Es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos. Se hace uso del mismo en la aplicación para establecer la conexión con el servidor que se desee administrar por lo que se hace una única instancia del objeto *SSHConection*, presente en el paquete *infrastructureCrosscutting*.

Conclusiones parciales

Con el estudio realizado sobre el servicio Cron se propone un módulo para HMAST que permite la planificación de tareas con 29 requisitos funcionales cada uno con su HU y 6 requisitos no funcionales. La arquitectura definida será N-Capas debido a que la herramienta principal emplea dicha arquitectura, lo que permitirá la homogeneidad del módulo con la herramienta base, tomada como estructura principal para la creación del diagrama de paquetes.

Capítulo 3: Implementación y pruebas del módulo de planificación de tareas para HMAST

En el presente capítulo se da a conocer el estándar de codificación a utilizar, el cual corresponde con el empleado por los módulos realizados para la herramienta con el objetivo de conservar la semejanza en la codificación. Se establece la estrategia de pruebas y se documenta la realización de las pruebas que se llevarán a cabo. Se muestra el diagrama de despliegue el cual facilita la comprensión de los componentes que conforman el sistema.

3.1. Estándar de codificación

El estándar de codificación es utilizado para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada (46)**.** El estándar empleado en el módulo se ajusta a las pautas definidas en el expediente de proyecto de HMAST.

- **Asignación de nombres:** Emplear descriptores en inglés. Evitar nombres largos y que difieran en una letra o en el uso de mayúsculas. Para nombrar las funciones y variables se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se utiliza la notación CamelCase[12].
- **Ficheros de código fuente:** Cada fichero contiene una única clase o interfaz. Si hay una clase privada o una interfaz asociada a una clase pública se puede poner en el mismo fichero. La clase pública debe ser la primera.
- **Indentación**: La unidad de indentación de bloques de sentencias son 4 espacios.
- **Comentarios**: Los comentarios deben añadir claridad al código. Deben contar el por qué y no el cómo. Deben ser concisos.
- **Declaraciones:** Se debe declarar cada variable en una línea distinta, de esta forma cada variable se puede comentar por separado.

[12] CamelCase es la práctica de escribir frases o palabras compuestas eliminando los espacios y poniendo en mayúscula la primera letra de cada palabra.

- **Continuidad de las líneas largas:** Cuando una sentencia no quepa en una única línea se debe fraccionar después de una coma, después de un operador y alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.
- **Longitud de la línea:** Limitar todas las líneas a un máximo de 120 caracteres.
- **Nombres de componentes:** Todos los paquetes comienzan con cu.uci.hmast.xxx.yyy.zzz.kkk

xxx → presentation, application, domain, persistence.

yyy → nombre del módulo (Cron).

zzz → elementos que pueden contener los componentes verticales (*entities, repositorys*).

kkk → clases o subpaquetes.

Una vez establecidos los estándares de codificación que serán utilizados se procede a plantear las estrategias de prueba que se aplicarán al módulo implementado.

3.2. Estrategia de pruebas

Las pruebas de *software* consisten en la dinámica de la verificación del comportamiento de un programa en un conjunto finito de casos de prueba, debidamente seleccionados, de por lo general infinitas ejecuciones de dominio, contra la del comportamiento esperado. 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 (47).

La metodología AUP-UCI desagrega las pruebas en tres disciplinas: internas, liberación y aceptación. Las pruebas de liberación son diseñadas y ejecutadas por una entidad certificadora externa, por lo que en el presente trabajo no son analizadas. La prueba de unidad analiza cada módulo individualmente, asegurando que funciona adecuadamente como una unidad, haciendo un uso intensivo del método de prueba de caja blanca y ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de errores. La prueba de integración se enfoca al diseño y la construcción de la arquitectura del *software*. La prueba de aceptación permite comprobar los requisitos establecidos durante el análisis de requisitos, comparándolos con el sistema que ha sido construido (48). En el caso de las pruebas de aceptación debido a que la aplicación solo se desarrolló hasta un 75%, no serán realizadas en su totalidad.

3.2.1. Prueba de unidad

Las pruebas de unidad son los procedimientos de pruebas locales a un módulo del sistema. Por definición, dichas pruebas cubren la funcionalidad propia del módulo tanto con una perspectiva de caja blanca como de caja negra (49). La prueba de unidad que se realiza hace uso del método de caja blanca y de la técnica del camino básico, esta técnica permite diseñar casos de prueba que garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa. Se desarrollan 25 requisitos funcionales a los que se les realiza la prueba de unidad. A continuación, se muestra el método *loadServiceConfiguration* (ver Tabla 9).

Tabla 7. Método loadServiceConfiguration para la prueba de caja blanca.

(1)	public Cron loadServiceConfiguration(LogicalServer server) throws InterruptedException, ENotFoundUrlRemoto, NoSuchAlgorithmException, IOException, JSchException, EInvalidEntity, SftpException, EFileWithIncorrectSettingsForAugeas, EErrorMessageOperatingSystem {
(2)	IdentityGenerator id = new IdentityGenerator();
(3)	UUID idCron = id.getNewUUID();
(4)	String baseDir = this.createRootFolderConfigurationModuleIfNotExist(server).getPath();
(5)	this.createPrincipalFileConfigIfNotExist(server);
(6)	Augeas augeas = new Augeas(baseDir, "", 0);
(7)	String pathBase = "/files" + serversAdministrationService. getConfigurationValueByKey (server.getId(),canonicalName,"rootFolderConfiguration") + serversAdministrationService. getConfigurationValueByKey(server.getId(),canonicalName, "nameFilePrincipalConfig");
(8)	if(!augeas.exists(pathBase)){
(9)	throw new EFileWithIncorrectSettingsForAugeas("Cron.errors.found.when.loading.file .incorrect.settings.for.augeas",serversAdministrationService. getConfigurationValueByKey (server.getId(),canonicalName,"rootFolderConfiguration") + serversAdministrationService. getConfigurationValueByKey(server.getId(),canonicalName,"nameFilePrincipalConfig")) ;
(10)	}
(11)	HourlyTasks hourlyTasks = this.getConfigurationHourlyTasks(server) ;
(12)	WeeklyTasks weeklyTasks = this.getConfigurationWeeklyTasks(server) ;
(13)	MonthlyTasks monthlyTasks = this.getConfigurationMonthlyTasks(server);

(14)	DailyTasks dailyTasks = this.getConfigurationDailyTasks(server) ;
(15)	List<String> entry = augeas.match(pathBase + "/entry");
(16)	List<Task> taskList = new LinkedList<>();
(17)	for (int i = 0; i < entry.size(); i++) {
(18)	String pathEntry = entry.get(i);
(19)	String entryDescription = "";
(20)	String entryDescription_ = augeas.get(pathEntry);
(21)	if (entryDescription_ != null && !entryDescription_.equals("")) {
(22)	entryDescription = entryDescription_;
(23)	}
(24)	if((!entryDescription.contains(serversAdministrationService. getConfigurationValueByKey (server.getId(),canonicalName,"directoryExecutionsHours"))) | |(!entryDescription.contains (serversAdministrationService.getConfigurationValueByKey (server.getId(),canonicalName, "directoryExecutionsDays"))) || (!entryDescription.contains(serversAdministrationService. getConfigurationValueByKey(server.getId(),canonicalName,"directoryExecutionsMonths"))) || (!entryDescription.contains(serversAdministrationService. getConfigurationValueByKey (server.getId(),canonicalName,"directoryExecutionsWeeks")))) {
(25)	Task task = getTask(augeas, pathEntry, entryDescription) ;
(26)	taskList.add(task);
(27)	}
(28)	}
(29)	Cron Cron = new Cron(idCron, taskList, weeklyTasks, hourlyTasks, dailyTasks, monthlyTasks) ;
(30)	return Cron;
(31)	}

Después de etiquetar las líneas de código se procede a representar el método en un grafo de flujo (ver Figura 5).

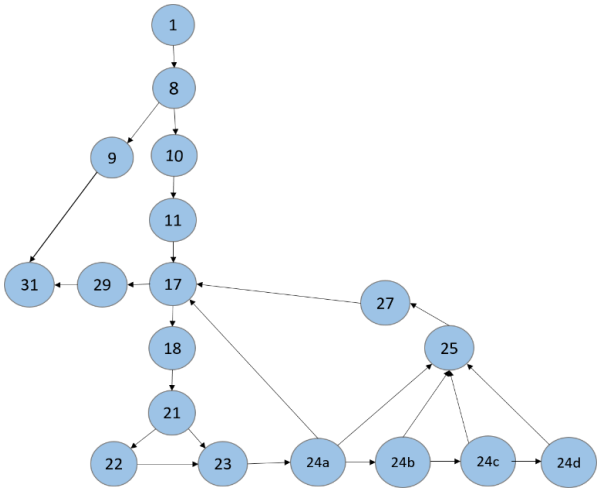


Figura 5. Grafo de Flujo.

Una vez obtenido el grafo se procede a calcular la complejidad ciclomática **V(G)**, que es una métrica de *software* que provee una medición cuantitativa de la complejidad lógica de un programa (50).

El grafo cuenta con 24 aristas y 18 nodos por lo que se plantea la siguiente fórmula.

V(G) = Cantidad_Aristas – Cantidad_Nodos + 2

V(G) = 24 – 18 + 2 = 8

Otra forma de calcular la complejidad ciclomática es utilizando los nodos predicados, los cuales no son más que cuando de un nodo emergen 2 o más nodos. Calculándose la complejidad ciclomática con la siguiente fórmula.

V(G) = Nodos_Predicados + 1

V(G) = 7 + 1 = 8.

Luego se determina un conjunto básico de caminos linealmente independientes. El valor de **V(G)** proporciona el número de caminos linealmente independientes de la estructura de control del programa (48). En este caso se especifican 8 caminos (ver Tabla 10).

Tabla 8. Caminos básicos.

No. Camino	Caminos
(1)	1-8-9-31
(2)	1-8-10-11-17-29-31
(3)	1-8-10-11-17-18-21-23-24a-17-29-31

(4)	1-8-10-11-17-18-21-22-23-24a-17-29-31
(5)	1-8-10-11-17-18-21-22-23-24a-25-27-29-31
(6)	1-8-10-11-17-18-21-22-23-24b-25-27-29-31
(7)	1-8-10-11-17-18-21-22-23-24c-25-27-29-31
(8)	1-8-10-11-17-18-21-22-23-24d-25-27-29-31

Se preparan los casos de prueba que forzarán la ejecución de cada camino del conjunto básico. Se deben escoger los datos de forma que las condiciones de los nodos predicado estén adecuadamente establecidas, con el fin de comprobar cada camino (48). A continuación, se encuentra cada caso de prueba diseñado (ver Tablas 11,12,13,14,15,16,17 y 18).

Tabla 9. Caso de prueba de unidad 1.

Casos de Prueba de Unidad:	
No. Ruta: 1	**Ruta:** 1-8-9-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que existe pero la ruta base a el archivo de configuración *crontab* no existe.	
Resultado esperado: Se muestra un mensaje de error en el que se comunica a el usuario que no se ha podido leer el archivo de configuración principal, *crontab*.	
Evaluación de la Prueba: Satisfactoria	

Tabla 10. Caso de prueba de unidad 2.

Casos de Prueba de Unidad:	
No. Ruta: 2	**Ruta:** 1-8-10-11-17-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que existe, además ya se encuentra guardado en el mapa y se ha obtenido el servicio por completo.	
Resultado esperado: Se retorna el servicio Cron.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 11. Caso de prueba de unidad 3.

Casos de Prueba de Unidad:	
No. Ruta: 3	**Ruta:** 1-8-10-11-17-18-21-23-24a-17-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que no contenía ejecuciones.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task*.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 12. Caso de prueba de unidad 4.

Casos de Prueba de Unidad:	
No. Ruta: 4	**Ruta:** 1-8-10-11-17-18-21-22-23-24a-17-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que no contenía ejecuciones y el archivo crontab va a ser leído por primera vez.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task*.	
Evaluación de la Prueba: Satisfactoria	

Tabla 13. Caso de prueba de unidad 5.

Casos de Prueba de Unidad:	
No. Ruta: 5	**Ruta:** 1-8-10-11-17-18-21-22-23-24a-25-27-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que contiene ejecuciones por hora.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task* y las ejecuciones por hora	
Evaluación de la Prueba: Satisfactoria	

Tabla 14. Caso de prueba de unidad 6.

Casos de Prueba de Unidad:	
No. Ruta: 6	**Ruta:** 1-8-10-11-17-18-21-22-23-24b-25-27-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que contiene ejecuciones diarias.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task* y las ejecuciones diarias.	
Evaluación de la Prueba: Satisfactoria	

Tabla 15. Caso de prueba de unidad 7.

Casos de Prueba de Unidad:	
No. Ruta: 7	**Ruta**1-8-10-11-17-18-21-22-23-24c-25-27-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que contiene ejecuciones semanales.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task* y las ejecuciones semanales.	
Evaluación de la Prueba: Satisfactoria	

Tabla 16. Caso de prueba de unidad 8.

Casos de Prueba de Unidad:	
No. Ruta: 8	**Ruta:** 1-8-10-11-17-18-21-22-23-24d-25-27-29-31
Nombre de la persona que realiza la prueba: Wilbia Esther Mariño Alcolea	
Descripción de la prueba: Obtener el servicio Cron.	
Entrada: Se envía como parámetro el uuid de un servidor que contiene ejecuciones mensuales.	
Resultado esperado: Se retorna el servicio Cron con una lista de tareas personalizadas de tipo *task* y las ejecuciones mensuales.	
Evaluación de la Prueba: Satisfactoria	

Como resultado de esta prueba se obtuvo que todas las sentencias de código se ejecutan al menos una vez, se detectaron 20 no conformidades en una primera iteración, las cuales todas fueron resueltas. Se encontraron 6 no conformidades en una segunda iteración a las que se le dieron solución y 0 no conformidades en la tercera iteración. Dichas no conformidades en su mayoría estuvieron relacionadas con faltas de ortografía en los mensajes mostrados, así como errores de redacción en los mismos.

3.2.2. Prueba de integración

Las pruebas de integración son una técnica para construir la arquitectura del *software*, verifican que los componentes de la aplicación funcionan correctamente actuando en conjunto. Este tipo de pruebas son dependientes del entorno en el que se ejecutan. Si fallan, puede ser porque el código esté bien, pero haya un cambio en el entorno (48) (51). Teniendo en cuenta que el módulo se debe añadir a una herramienta base y la arquitectura empleada es por capas, se recurre a una estrategia de integración ascendente, donde los componentes se integran de abajo hacia arriba. En el contexto de la prueba de integración ascendente se realiza la prueba de regresión que permite ejecutar nuevamente el mismo subconjunto de pruebas que ya se ha aplicado para asegurar que los cambios no han propagado efectos colaterales indeseables (48). Se realiza esta prueba hasta un 75% debido a que el desarrollo del módulo propuesto solo se implementó hasta este %.

3.2.3. Prueba de aceptación

Las pruebas de aceptación es el proceso de revisión que verifica que el sistema producido cumple con las especificaciones y logra su cometido, por lo que se realizan pruebas de aceptación. Como técnica se escoge las pruebas alfas esta se lleva a cabo por un cliente en el lugar de desarrollo. Estas pruebas se llevan a cabo en un entorno controlado y se usa el software de forma natural con el desarrollador como observador del usuario.

Para que tengan validez se debe crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto se procede a realizar las pruebas y a documentar los resultados. El resultado de las pruebas de aceptación se muestra en el Anexo 2, donde fue emitida el acta de aceptación por el cliente.

3.3. Diagrama de despliegue

Consiste en un diagrama estructurado que muestra la arquitectura del sistema desde el punto de vista de distribución de los artefactos del *software* en los destinos de despliegue. La herramienta HMAST se ejecuta sobre un servidor web Tomcat sobre el sistema operativo GNU/Linux Nova. El usuario accede a HMAST mediante un navegador web y empleando conexiones seguras con el protocolo HTTPS. A través del módulo Planificador de Tareas de HMAST el usuario accede al servidor controlador de dominio y realiza la operación de administración. A continuación, se presenta la distribución física del módulo (ver Figura 7).

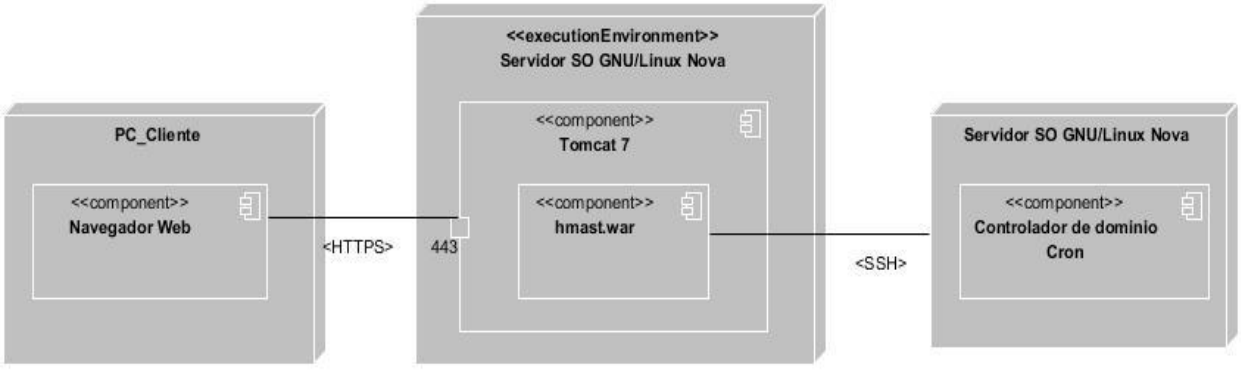


Figura 5. Diagrama de despliegue del módulo.

Conclusiones parciales

Durante la etapa de implementación el uso del estándar de codificación definido para la herramienta HMAST permitió desarrollar un código reutilizable de fácil comprensión para la implementación de nuevas funcionalidades y mejoras a realizar en futuras versiones del módulo. Una vez desarrolladas las funcionalidades del *software*, se documentó y ejecutó la prueba de unidad basada en el método de caja blanca, obteniéndose resultados satisfactorios. De esta forma se detectaron no conformidades que fueron erradicadas en su totalidad; luego de solucionadas estas, se obtiene un sistema que cumple con las funcionalidades definidas al inicio de la investigación

Conclusiones Generales

Al finalizar la investigación realizada se concluye que:

- El análisis de aspectos teóricos asociados a la investigación permitió tener un mayor conocimiento del servicio de planificación de tareas.
- La caracterización de las herramientas existentes para la administración del servicio Cron permitió concluir que estas no pueden ser integradas con HMAST.
- La correcta utilización de las herramientas, lenguajes, tecnologías y de la metodología AUP-UCI, hicieron posible la obtención de un diseño e implementación apropiados para el módulo desarrollado.
- El empleo de la arquitectura N-Capas orientada al Dominio permitió la reutilización de código e integración a HMAST.

Recomendaciones

Como recomendaciones de la presente investigación se plantea:

- La implementación de los requisitos funcionales 22, 23,24 y 25 pertenecientes a la gestión de los grupos de tareas.
- La implementación de nuevas funcionalidades que permitan gestionar los archivos de configuración Cron.deny y Cron.allow.

Referencias Bibliográficas

1. INFORMATICA, KMICOS UNIDOS X INFORMÄTICA. La Telemática. [En línea] 20 de enero de 2012. [Citado el: 19 de octubre de 2016.] http://telematimika.blogspot.com/.

2. Observatorio Tecnologíco. *Automatización de tareas en sistemas GNU/Linux.* [En línea] 8 de febrero de 2008. [Citado el: 2 de mayo de 2017.] http://recursostic.educacion.es/observatorio/web/en/software/software-general/558-raul-juncos.

3. Villazón, Yoandy Pérez. *Estrategia para la migración a aplicaciones de código abierto.* La Habana : s.n., diciembre, 2015.

4. Pérez Villazón, Yoandy y Paumier Samón, Ramón . *Guía Cubana de Migración a Software Libre.* 2009.

5. Informatica sin limites. [En línea] abril de 2011. [Citado el: 2 de mayo de 2017.] https://informatica-sinlimites.blogspot.com/2011/04/que-son-las-tareas-programadas.html.

6. [En línea] 2 de octubre de 2012. [Citado el: 1 de junio de 2017.] https://blog.desdelinux.net/ejecuta-un-comando-a-la-hora-que-quieras-con-at/.

7. [En línea] 3 de enero de 2011. [Citado el: 1 de junio de 2017.] https://mislinuxapps.wordpress.com/2011/01/03/programando-tareas-en-linux/.

8. Debian. *Debian 8. El manual del Administrador de Debian.* Primera. 2015. ISBN: 979-10-91414-09-8.

9. Ministerio de Educación, Cultura y Deporte España. Observatorio Tecnológico. [En línea] 8 de febrero de 2008. [Citado el: 21 de marzo de 2017.] http://recursostic.educacion.es/observatorio/web/en/software/software-general/558-raul-juncos.

10. Roland Mas, Raphaël Hertzog. *Debian 8.El manual del Administrador de Debian.Debian Jessie desde el descubrimiento a la maestría.* 2015. ISBN: 979-10-91414-09-8.

11. Durán, Sergio González. LinuxTotal.com.mx . [En línea] 2015. [Citado el: 28 de noviembre de 2016.] http://www.linuxtotal.com.mx/?cont=info_admon_006.

12. Oracle. *Guía de administración del sistema: administración avanzada.* [En línea] 2011. [Citado el: 7 de diciembre de 2016.] https://docs.oracle.com/cd/E24842_01/html/E23086/index.html.

13. Castillo Arbelo, Raidiel y Acosta Soria, Pablo. *Herramienta para la Migración y Administración de Servidores (HMAST).* La Habana : Universidad de las Ciencias Informáticas, 2012.

14. Castillo Ramos, Leosdanis y Quezada Arévalo, Gustavo . *Módulo para la gestión de copias de seguridad desde la Herramienta para la Migración y Administración de los Servicios Telemáticos.* La Habana : Universidad de las Ciencias Informáticas, 2014.

15. Tasé, Alexander Pérez. *Módulo de administración del servicio Proxy para HMAST.* La Habana : Universidad de las Ciencias Informáticas, 2013.

16. González Santiago, Felipe y Vera González, Yosel Lázaro. *Módulo de HMAST para la Administración y Migración hacia Samba4 del servicio directorio activo.* La Habana : Universidad de las Ciencias Informáticas , 2015.

17. © Gaute Hope. Gnome schedule. [En línea] 2009. [Citado el: 7 de diciembre de 2016.] http://gnome-schedule.sourceforge.net/.

18. Webmin. [En línea] 2016. [Citado el: 7 de diciembre de 2016.] http://www.webmin.com/intro.html.

19. Ubuntu es. [En línea] 6 de marzo de 2010. [Citado el: 7 de diciembre de 2016.] http://www.ubuntu-es.org/node/129411.

20. Webmin. [En línea] 2 de septiembre de 2015. [Citado el: 7 de diciembre de 2016.] http://doxfer.webmin.com/Webmin/Scheduled_Cron_Jobs.

21. Libre, Centro de Software. *Administración de un servidor de almacenamiento conectado a la red con Nova NAS. Manual de Referencia.* La Habana : s.n., 2015.

22. Tamayo, Yoel Miyares. *Administración del servicio antivirus desde la Herramienta para la Migración y Administración de Servicios Telemáticos (HMAST).* La Habana : Universidad de las Ciencias Informáticas, 2014.

23. Informáticas, Universidad de las Ciencia. *Metodología de desarrollo para la Actividad productiva de la UCI.* 2015.

24. Lenguaje de programación/ definición de Lenguaje de programación. [En línea] [Citado el: 4 de diciembre de 2016.] http://www.diclib.com/lenguaje%20de%20programaci%C3%B3n/show/es/es_wiki_10/74813.

25. Sierra, Katy y Bates, Bert. *Java Head First.* s.l. : 2da Edición. 677 p.

26. Cwalina, Krysztof y Abrams, Brad. *Framework Design Guidelines.* octubre, 2008. 2da Edición.

27. Spring Framework Reference Documentation. [En línea] [Citado el: 4 de diciembre de 2016.] http://docs.spring.io/spring/docs/4.1.0.BUILD-SNAPSHOT/spring-frameworkreference/htmlsingle/..

28. Fergarciac. [En línea] 25 de enero de 2013. [Citado el: 5 de diciembre de 2016.] https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

29. Villegas, David López. Academia Android. [En línea] 17 de mayo de 2014. [Citado el: 23 de febrero de 2017.] http://academiaandroid.com/ide-android-intellij-android-studio-aide/.
30. CCM. [En línea] diciembre de 2016. [Citado el: 5 de diciembre de 2016.] http://es.ccm.net/contents/222-ingenieria-de-software-asistida-por-ordenador-case.
31. Software.com.ar. [En línea] [Citado el: 5 de diciembre de 2016.] http://www.software.com.ar/p/visual-paradigm-para-uml#product-description.
32. Aprenderaproramar.com. [En línea] 2016. [Citado el: 8 de diciembre de 2016.] http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=688:ique-es-y-para-que-sirve-uml-versiones-de-uml-lenguaje-unificado-de-modelado-tipos-de-diagramas-uml&catid=46:lenguajes-y-entornos&Itemid=163.
33. Booch, Gandy, Rumbaugh, Jim y Jacobobson, Ivar. *UML. Lenguaje Unificado de modelado.*
34. Diagramas UML. Catedra de Proyecto. [En línea] www.teatroabadia.com/es/uploads/documentos/iagramas_del_uml.pdf.
35. Augeas. [En línea] [Citado el: 31 de mayo de 2017.] http://augeas.net.
36. Straub, Ben y Chacon, Scoot. *Pro Git.* 2. 2014. ISBN-13: 978-1484200773.
37. Slide Share. [En línea] 25 de enero de 2011. [Citado el: 15 de junio de 2017.] https://es.slideshare.net/juanchenao/tipos-de-requisitos.
38. Sommerville, Ian. *Ingeniería de Software.* 7m edición. 2005.
39. PMOinformatica.com. *La oficina de proyectos de informática. La web sobre gerencia de proyectos de informática, software y tecnología.* [En línea] 1 de junio de 2015. [Citado el: 14 de febrero de 2017.] http://www.pmoinformatica.com/2015/05/historias-de-usuario-ejemplos.html.
40. de la Torre LLorente, César, y otros. *Guía de Arquitectura N-Capas orientada al Dominio con . NET 4.0.* 2010. ISBN: 978-84-936696-3-8.
41. Gutierrez, Demián. *UML. Diagrama de Paquetes (UML Ilustrado).* Venezuela : s.n., 2009.
42. GENBETA: dev. [En línea] 14 de julio de 2014. [Citado el: 22 de febrero de 2017.] https://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.
43. Usaola, Macario Polo. *Patrones GRASP.*
44. Blog de Julio Pari. [En línea] 2017. [Citado el: 15 de 5 de 2017.] http://blog.juliopari.com/alta-cohesion-y-bajo-acoplamiento-diseno-de-software/.

45. Uno Informatica. [En línea] 16 de julio de 2015. [Citado el: 14 de junio de 2017.] https://unoinformatica.wordpress.com/2015/07/16/patronesdediseno/.
46. Microsoft. msdn.microsoft.com. [En línea] [Citado el: 19 de marzo de 2017.] http://msdn.microsoft.com/es-es/library/aa291591(v=vs.71).aspx..
47. Gutiérrez, Javier. *Introducción a el proceso de pruebas.* España : s.n.
48. Pressman, Roger. S. *Ingeniería de software. Un enfoque práctico.* Quinta .
49. Tecnónología y Synergix. [En línea] 15 de marzo de 2008. [Citado el: 23 de marzo de 2017.] https://synergix.wordpress.com/2008/03/15/definimos-pruebas-de-unidad-como/.
50. GAVIRIA, BEATRIZ FLORIAN. *TÉCNICAS DE PRUEBAS DE SOFTWARE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN.* UNIVERSIDAD DEL VALLE. s.l. : SEMESTRE 2013A , SEMESTRE 2013A . Clase.
51. Ana M. del Carmen García Oterino . Javierganzas,com. [En línea] http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.html.

Anexos:

Anexo 1: Historias de Usuario

1- Reiniciar el servicio Cron.

Número: 2	**Nombre del requisito:** Reiniciar el servicio Cron.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 horas
Descripción: Permite reiniciar el servicio Cron. La funcionalidad inicia cuando el usuario selecciona el botón de reiniciar el servicio. Para esto se ejecutará el comando *$ sudo service cron restart* en la PC servidora.	
Observaciones: El servicio debe de estar en ejecución.	
Prototipo elemental de interfaz gráfica de usuario: 	

3- Detener el servicio Cron.

Número: 3	**Nombre del requisito:** Detener el servicio Cron.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 5 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 5 horas
Descripción: Permite detener el servicio Cron. La funcionalidad inicia cuando el usuario selecciona el botón de detener el servicio. Para esto se ejecutará el comando *$ sudo service cron stopt* en la PC servidora.	
Observaciones: El servicio debe de estar en ejecución.	
Prototipo elemental de interfaz gráfica de usuario: 	

4- Aplicar cambios al servidor.

Número: 4	**Nombre del requisito:** Aplicar cambios al servidor.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 10 horas

Descripción: Permite aplicar los cambios que se han realizado, al servidor.

La funcionalidad inicia cuando el usuario presiona el botón *Aplicar* y esta funcionalidad presenta dos variantes.

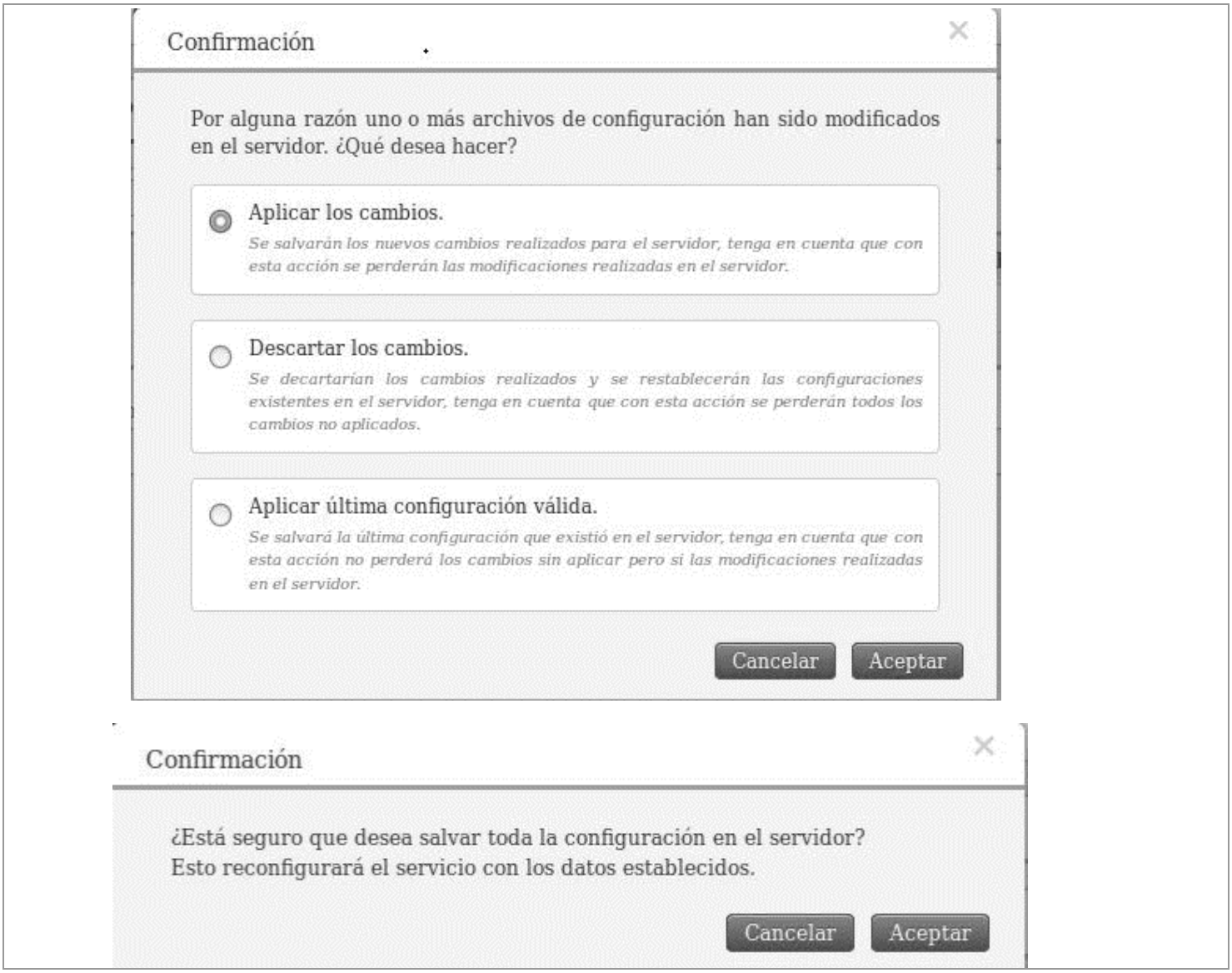
1- Existen cambios por otro usuario en el servidor que se está administrando. Para esto se muestra una interfaz con tres opciones para que el usuario seleccione la que desea. A continuación, se describen estas opciones:

- **Aplicar cambios:** Se salvarán los nuevos cambios realizados para el servidor y se perderán las configuraciones que fueron realizadas por otro usuario en el servidor.
- **Descartar cambios:** Se descartan los cambios realizados y se restablecen las configuraciones existentes en el servidor, al seleccionar esta opción se perderán todos los cambios que no han sido aplicados al servidor.
- **Aplicar última configuración válida:** Se salvará la última configuración que existió en el servidor, al seleccionar esta opción no se perderán los cambios sin aplicar, pero sí los cambios que fueron realizados en el servidor por un tercer usuario.

2- No existen cambios en el servidor por otro usuario, por lo que se muestra un mensaje de confirmación para reconfigurar el servicio en el servidor.

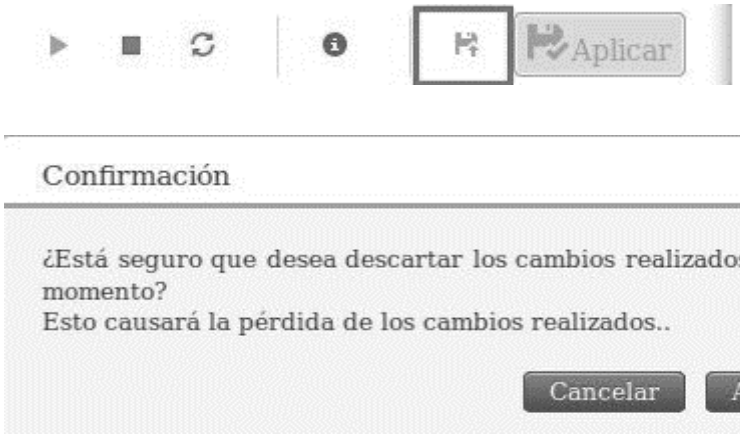
Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



5- Descartar cambios en el servidor.

Número: 5	**Nombre del requisito:** Descartar cambios realizados localmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 10 horas
Descripción: Permite descartar los cambios que fueron realizados localmente.	

La funcionalidad inicia cuando el usuario presiona el botón *Descartar* y automáticamente se muestra un cartel de confirmación para completar la acción, el usuario aceptará y se mostrará un mensaje que comunica que la acción se ha realizado correctamente. Esta funcionalidad permite descartar los cambios que se han realizado hasta este momento.
Observaciones:
Prototipo elemental de interfaz gráfica de usuario: 

7- Listar tareas personalizadas.

Número: 7	**Nombre del requisito:** Listar tareas personalizadas.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 24 horas
Descripción: Permite listar las tareas personalizadas. La funcionalidad inicia cuando el usuario selecciona la opción de *Tareas Personalizadas*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Se muestra una interfaz en la que se listan todas las tareas personalizadas que se encuentran en el servidor, organizadas por columnas donde la primera corresponderá al usuario encargado de ejecutar la tarea, seguido por las columnas de días del mes, días de la semana, horas, minutos y mes en los cuales se ejecutará la tarea personalizada.	
Observaciones:	

Prototipo elemental de interfaz gráfica de usuario:

8- Editar tarea personalizada.

Número: 8	**Nombre del requisito:** Editar tarea personalizada.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 24 horas

Descripción: Permite editar las tareas personalizadas.

La funcionalidad inicia cuando el usuario selecciona la opción *Tareas Personalizadas*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página y antes de selecciona el botón *Editar* que se encuentra en la parte superior derecha, tendrá que seleccionar una tarea personalizada. Al instante aparecerá una interfaz con los atributos que componen a una tarea personalizada, los cuales se describen a continuación, para que el usuario los modifique.

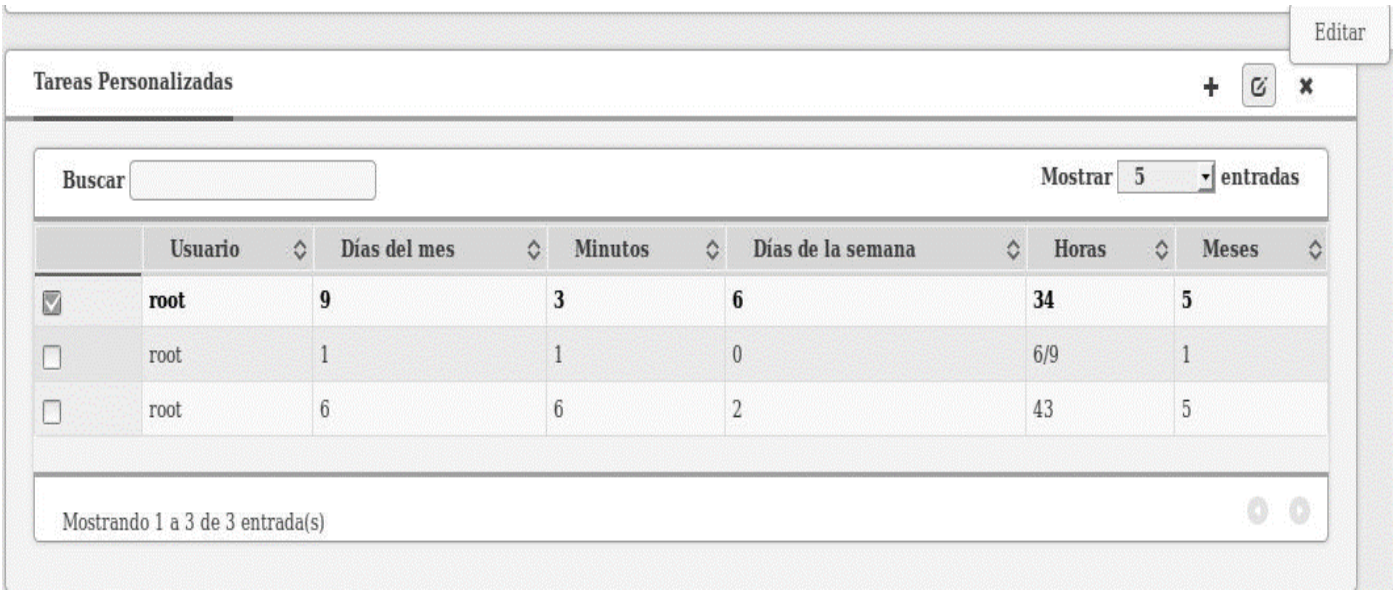
- **Minutos:** son los minutos en los cuales se va a ejecutar la tarea. Permite caracteres como (*), (,), (/), (-) y números y tiene un rango de 0-59. Es un campo de texto y es obligatorio.
- **Hora:** es la hora en que se ejecutará la tarea, este tiene un formato de 24 horas, son números enteros y tiene un rango de 0-23. Permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.
- **Día del mes:** es el día del mes en el cual se va a ejecutar la tarea, presenta un rango de 1-31, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.
- **Mes:** es el mes en el cual se va a ejecutar la tarea, presenta un rango de 1-12, son números enteros y

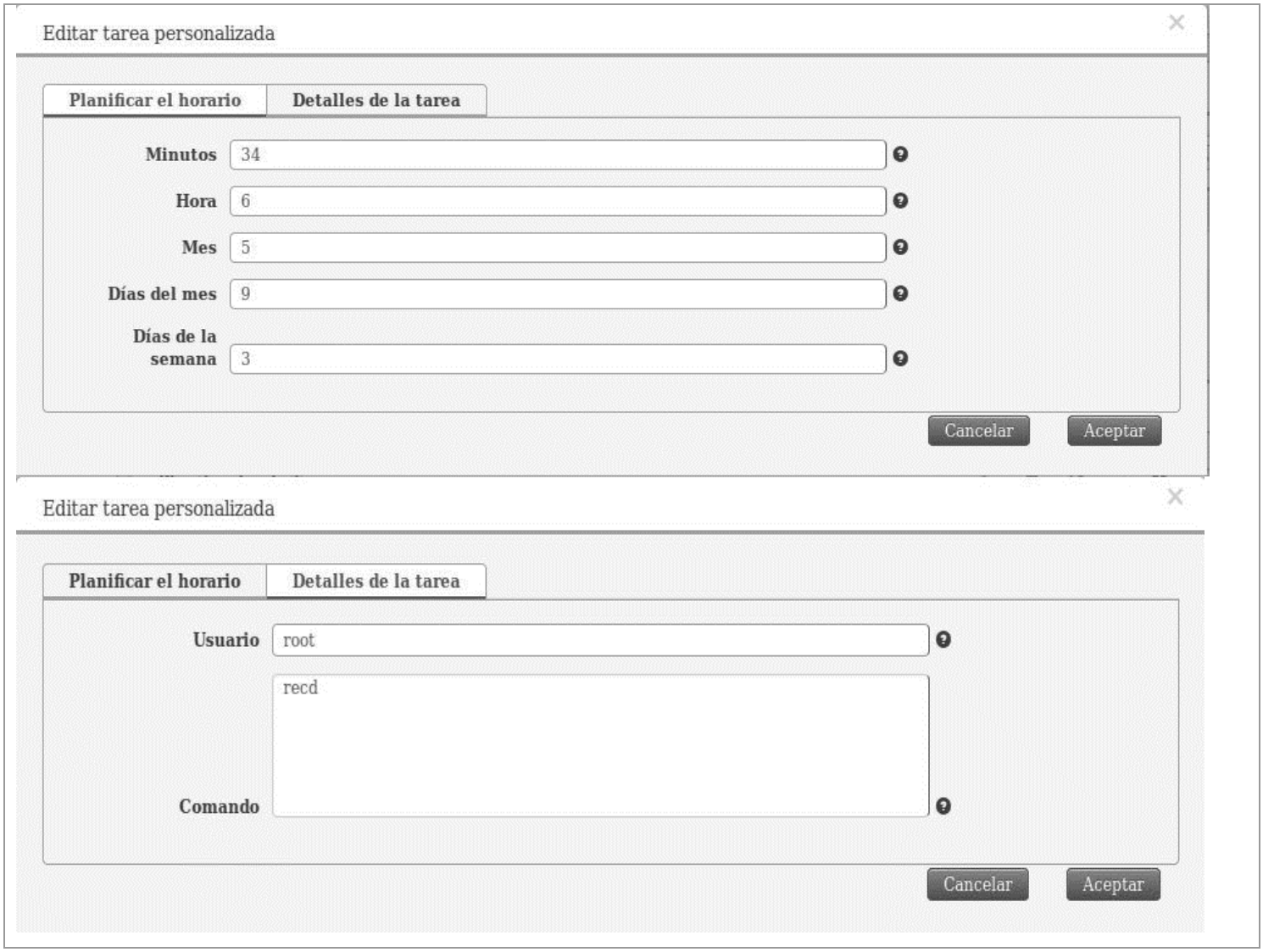
permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.

- **Día de la Semana:** es el día de la semana en el cual se va a ejecutar la tarea, presenta un rango de 0-7 siendo 0 y 7 valores correspondientes al domingo, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.
- **Usuario:** es el usuario que va a ejecutar la tarea que se está creando. Es un campo de texto y es obligatorio. Permite hasta 255 caracteres de a-z.
- **Comando:** es el comando que va a ser ejecutado. Es un campo de texto y es obligatorio. Permite hasta 1037 caracteres.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:





9- Eliminar tarea personalizada.

Número: 9	**Nombre del requisito:** Eliminar tareas personalizadas.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 24 horas
Riesgo en Desarrollo: Alto	**Tiempo Real:** 24 horas
Descripción: Permite eliminar las tareas personalizadas. La funcionalidad inicia cuando el usuario selecciona la opción *Tareas Personalizadas*, se listan todas las	

tareas personalizadas que se encuentran en el servidor, organizadas por columnas. El usuario seleccionará la tarea o las tareas deseadas y accederá al botón *Eliminar* que se encuentra en la parte superior derecha. Luego se muestra una interfaz en la que se mostrará un mensaje de confirmación para la acción a ejecutar.
Observaciones:
Prototipo elemental de interfaz gráfica de usuario:

10- Listar ejecuciones realizadas a cada hora.

Número: 10	**Nombre del requisito:** Listar ejecuciones realizadas a cada hora.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite listar los scripts que serán ejecutados a cada hora. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones por hora*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos	

pestañas, seleccionará la pestaña *Ejecuciones*, a continuación se mostrará un listado con los nombres y la ruta de los scripts que se encuentran en el directorio */etc/cron.hourly/* los cuales son las ejecuciones realizadas por hora que se encuentran en el servidor.
Observaciones:
Prototipo elemental de interfaz gráfica de usuario: 

11- Adicionar ejecuciones realizadas a cada hora.

Numero: 11	**Nombre del requisito:** Adicionar ejecución realizada a cada hora.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite adicionar un script que será ejecutado a cada hora. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones por hora*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el botón de adicionar que se encuentra en la parte superior derecha y después se mostrará una interfaz que permite adicionar un nuevo script para que se ejecute a cada hora. Este nuevo script de añadirá al directorio */etc/cron.hourly/.* La interfaz estará compuesta por los siguientes atributos.	

- **Nombre:** Nombre que tienes el nuevo script a adicionar. Es un campo que permite cargar el nuevo script, del cual solo se mostrar el nombre, es un campo obligatorio.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:

12- Eliminar ejecuciones realizadas a cada hora.

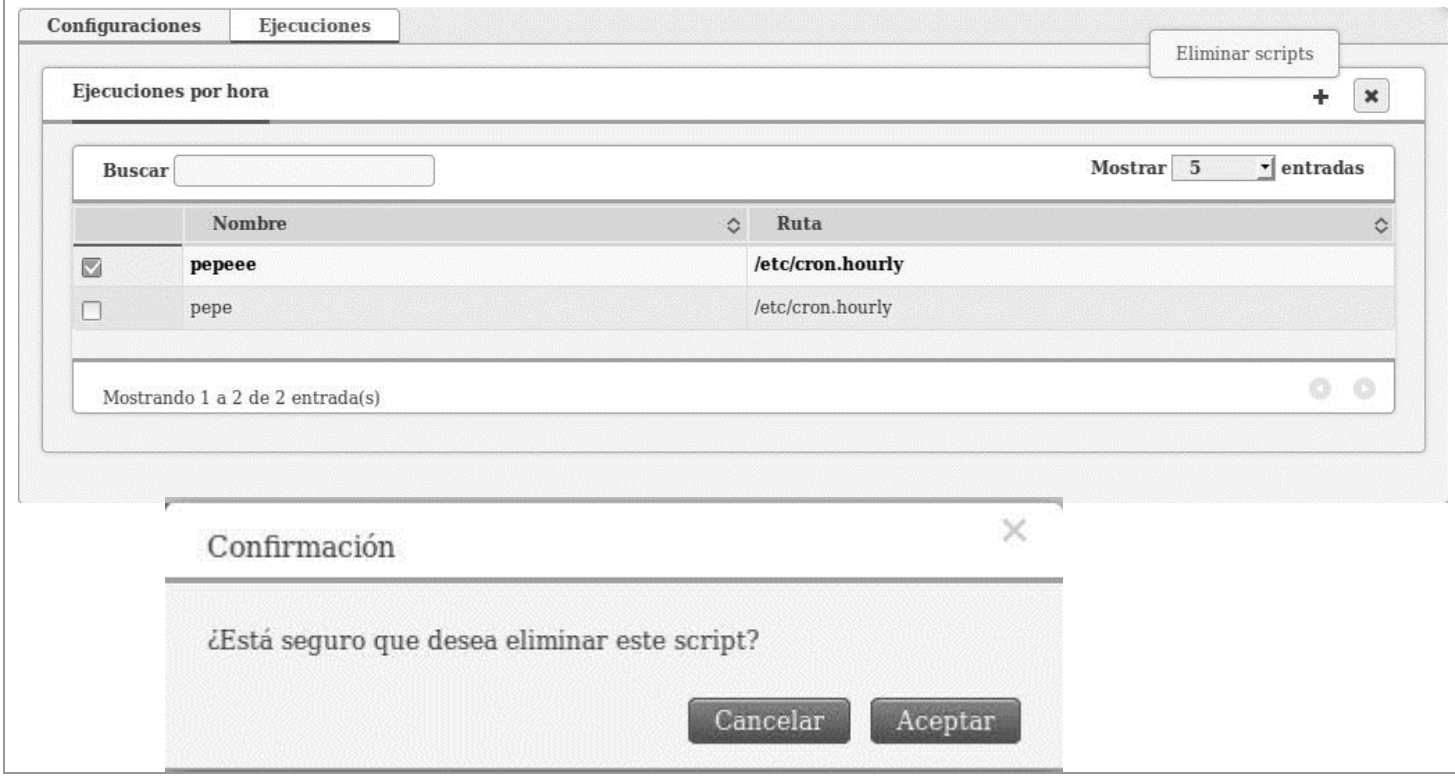
Numero: 12	**Nombre del requisito:** Eliminar ejecuciones realizadas a cada hora.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas

Descripción: Permite eliminar scripts que se ejecuten a cada hora.

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones por hora*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el usuario selecciona el o los scripts que se deseen eliminar, seleccionará el botón de eliminar que se encuentra en la parte superior derecha y después se mostrará una interfaz para que el usuario confirme la acción que va a realizar. Esta acción permite eliminar un script del directorio */etc/cron.hourly/.*

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



13- Editar tiempo en que se realizan las ejecuciones por hora.

Número: 13	**Nombre del requisito:** Editar tiempo en que se realizan las ejecuciones por hora.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas

Descripción: Permite editar el tiempo en que se realizan las ejecuciones por hora.

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones por hora*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Configuraciones*. En esta se encuentra el atributo que puede ser modificado en este tipo de ejecución con el valor que este presenta en el archivo */etc/crontab*, a continuación, se describe este atributo.

- **Minutos:** son los minutos en los cuales se va a ejecutar la tarea. Permite caracteres como (*), (,), (/), (-) y números y tiene un rango de 0-59. Es un campo de texto y es obligatorio.

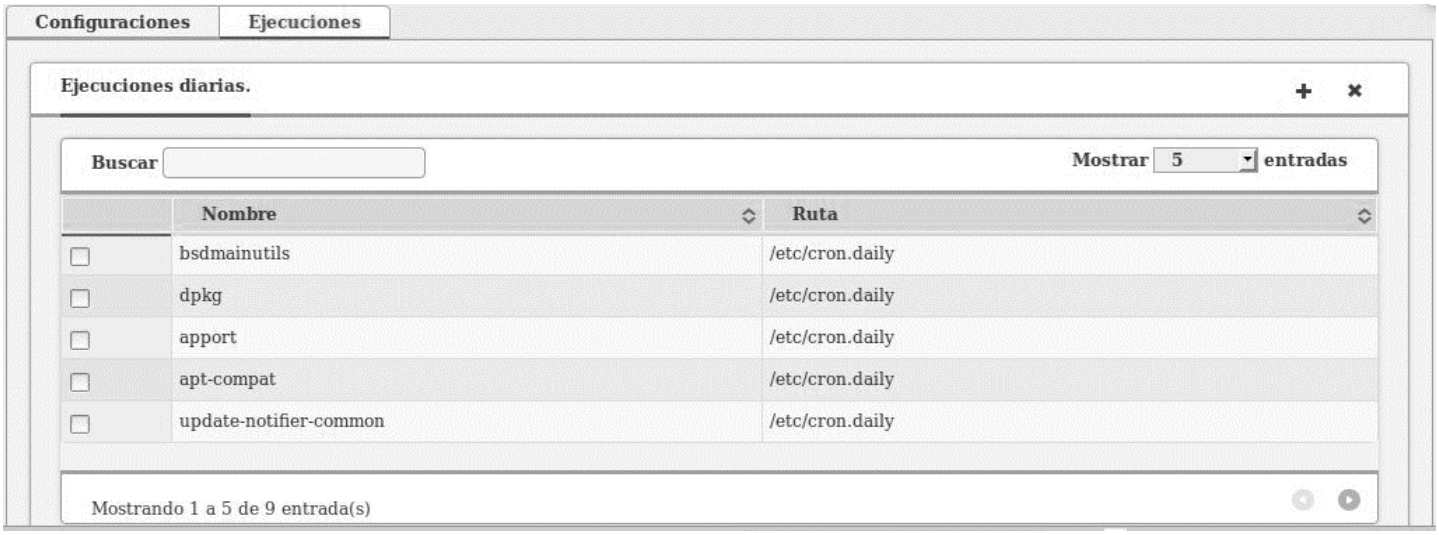
Observaciones:

Prototipo elemental de interfaz gráfica de usuario:

14- Listar ejecuciones realizadas diariamente.

Número: 14	**Nombre del requisito:** Listar ejecuciones realizadas diariamente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas

Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite listar los scripts que serán ejecutados diariamente. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones diarias*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación se mostrará un listado con los nombres y la ruta de los scripts que se encuentran en el directorio */etc/cron.daily/* los cuales son las ejecuciones realizadas diariamente que se encuentran en el servidor.	
Observaciones:	
Prototipo elemental de interfaz gráfica de usuario:	



15- Adicionar ejecución realizada diariamente.

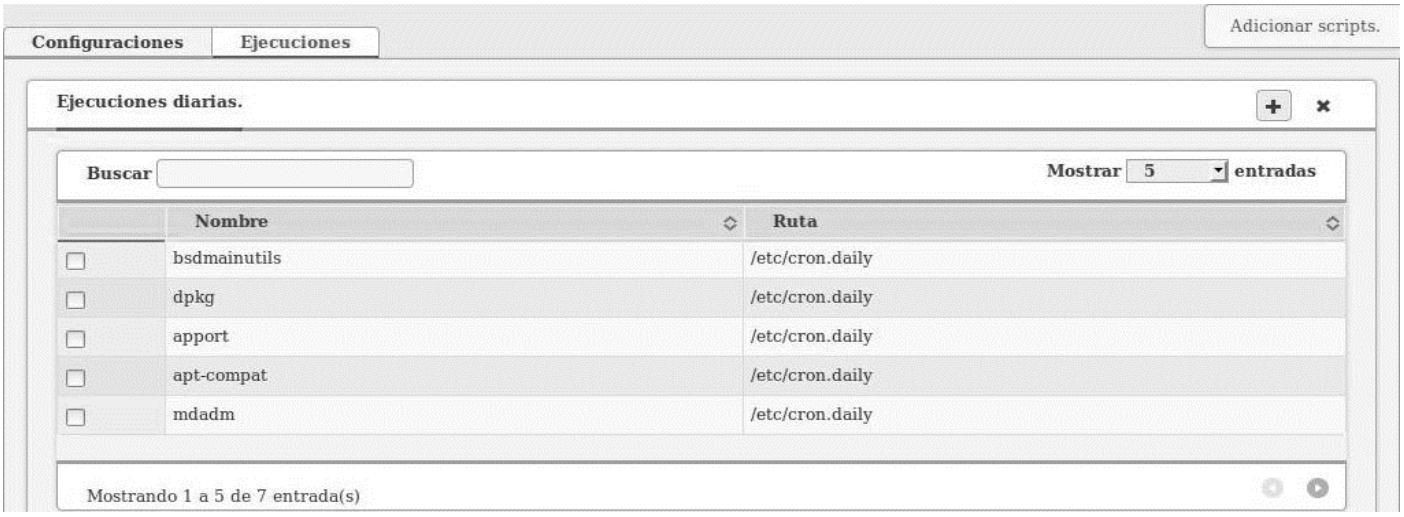
Numero: 15	**Nombre del requisito:** Adicionar ejecución realizada diariamente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite adicionar un script que será ejecutado diariamente. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones diarias*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos	

pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el botón de adicionar que se encuentra en la parte superior derecha y después se mostrará una interfaz que permite adicionar un nuevo script para que se ejecute diariamente. Este nuevo script de añadirá al directorio */etc/cron.daily/.* La interfaz estará compuesta por los siguientes atributos.

- **Nombre:** Nombre que tienes el nuevo script a adicionar. Es un campo que permite cargar el nuevo script, del cual solo se mostrar el nombre, es un campo obligatorio.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



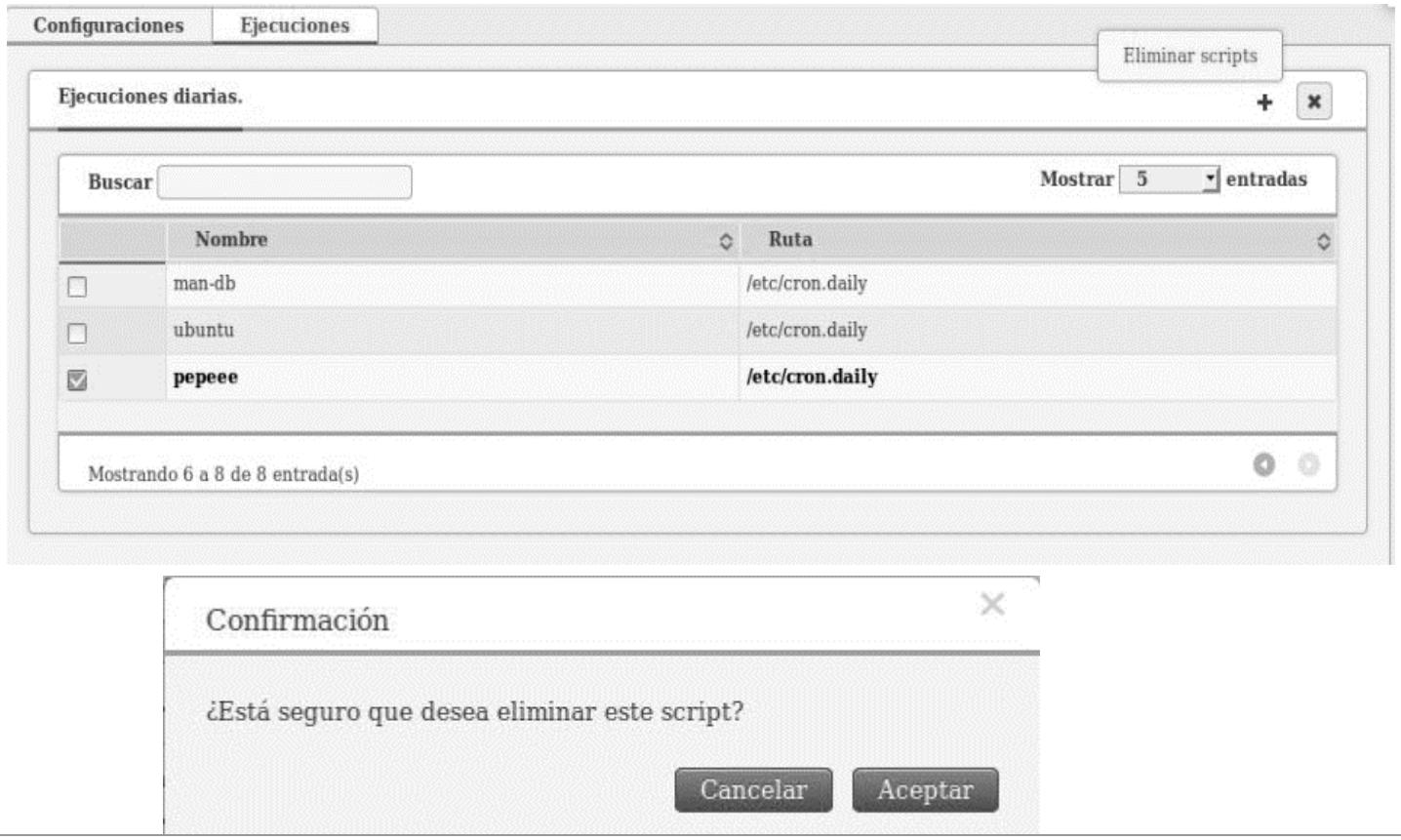
La segunda interfaz de este requisito corresponde con la segunda interfaz de la historia de usuario de adicionar ejecución realizada a cada hora.

16- Eliminar ejecuciones realizadas diariamente.

Numero: 16	**Nombre del requisito:** Eliminar ejecuciones realizadas diariamente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas

Descripción: Permite eliminar scripts que se ejecuten diariamente.

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones diarias*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos

pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el usuario después seleccionar el o los scripts que se deseen eliminar, seleccionará el botón de eliminar que se encuentra en la parte superior derecha y después se mostrará una interfaz para que el usuario confirme la acción que va a realizar. Esta acción permite eliminar un script del directorio */etc/cron.daily/.*
Observaciones:
Prototipo elemental de interfaz gráfica de usuario: 

17- Editar tiempo en que se realizan las ejecuciones diarias.

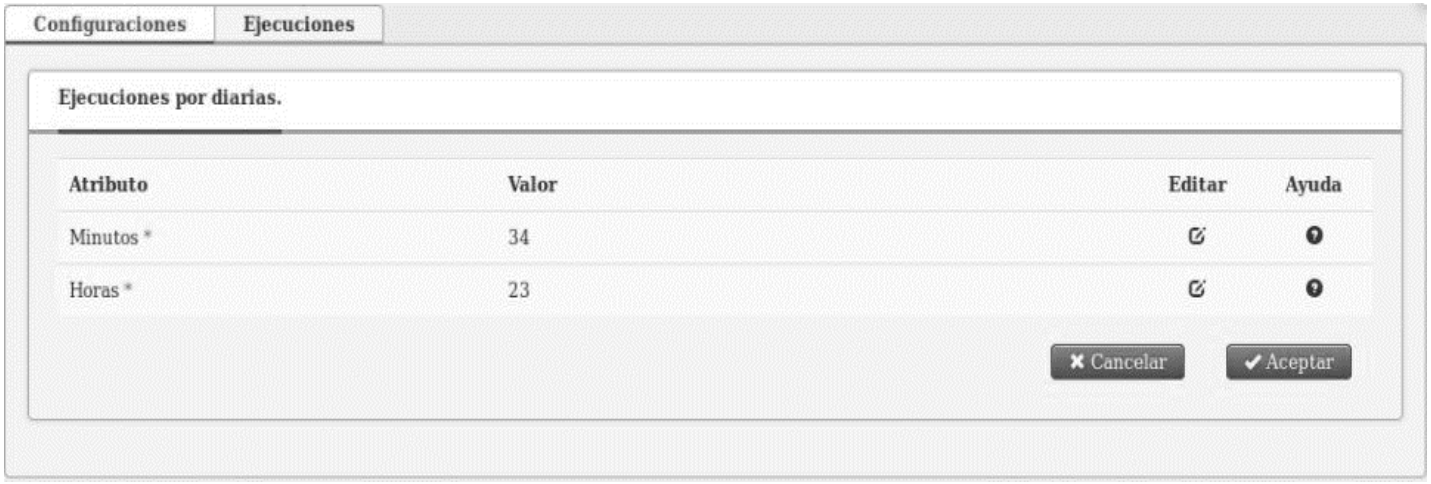
Número: 17	**Nombre del requisito:** Editar tiempo en que se realizan las ejecuciones diarias.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite editar el tiempo en que se realizan las ejecuciones diarias.	

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones diarias*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Configuraciones*. En esta se encuentran los atributos que pueden ser modificados en este tipo de ejecución con los valores que estos representan en el archivo */etc/crontab*, a continuación, se describen estos atributos.

- **Minutos:** son los minutos en los cuales se va a ejecutar la tarea. Permite caracteres como (*), (,), (/), (-) y números y tiene un rango de 0-59. Es un campo de texto y es obligatorio.
- **Hora:** es la hora en que se ejecutará la tarea, este tiene un formato de 24 horas, son números enteros y tiene un rango de 0-23. Permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



18- Listar ejecuciones realizadas semanalmente.

Número: 18	**Nombre del requisito:** Listar ejecuciones realizadas semanalmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas

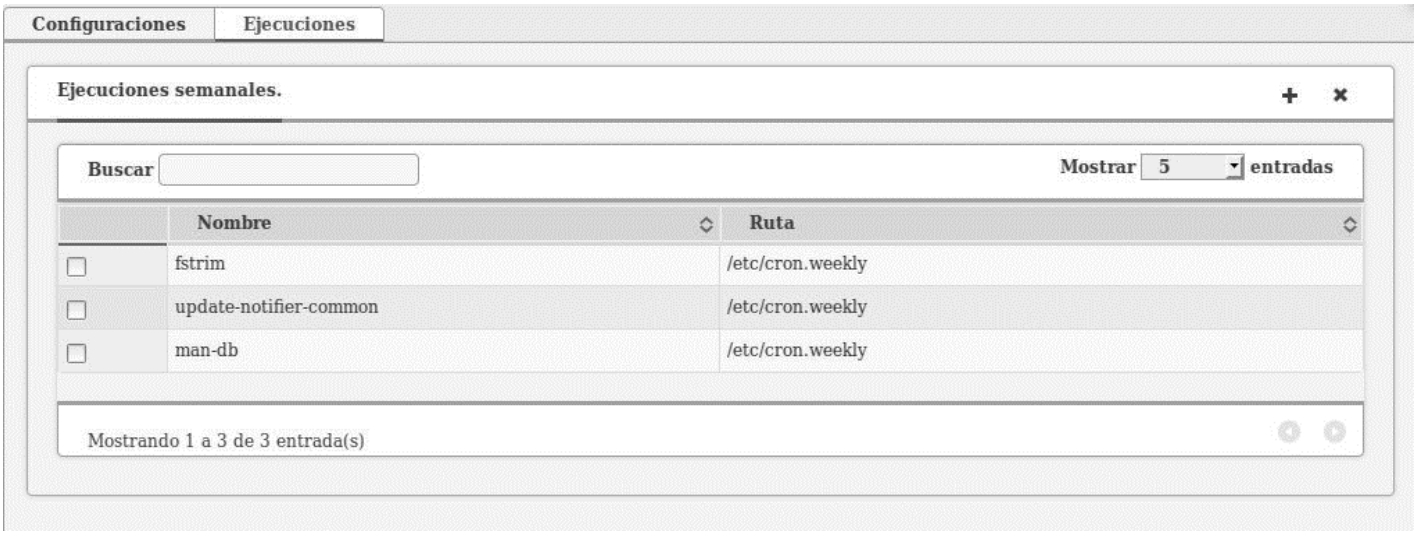
Descripción: Permite listar los scripts que serán ejecutados semanalmente.

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones semanales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos

pestañas, seleccionará la pestaña *Ejecuciones*, a continuación se mostrará un listado con los nombres y la ruta de los scripts que se encuentran en el directorio */etc/cron.weekly/* los cuales son las ejecuciones realizadas semanalmente que se encuentran en el servidor.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



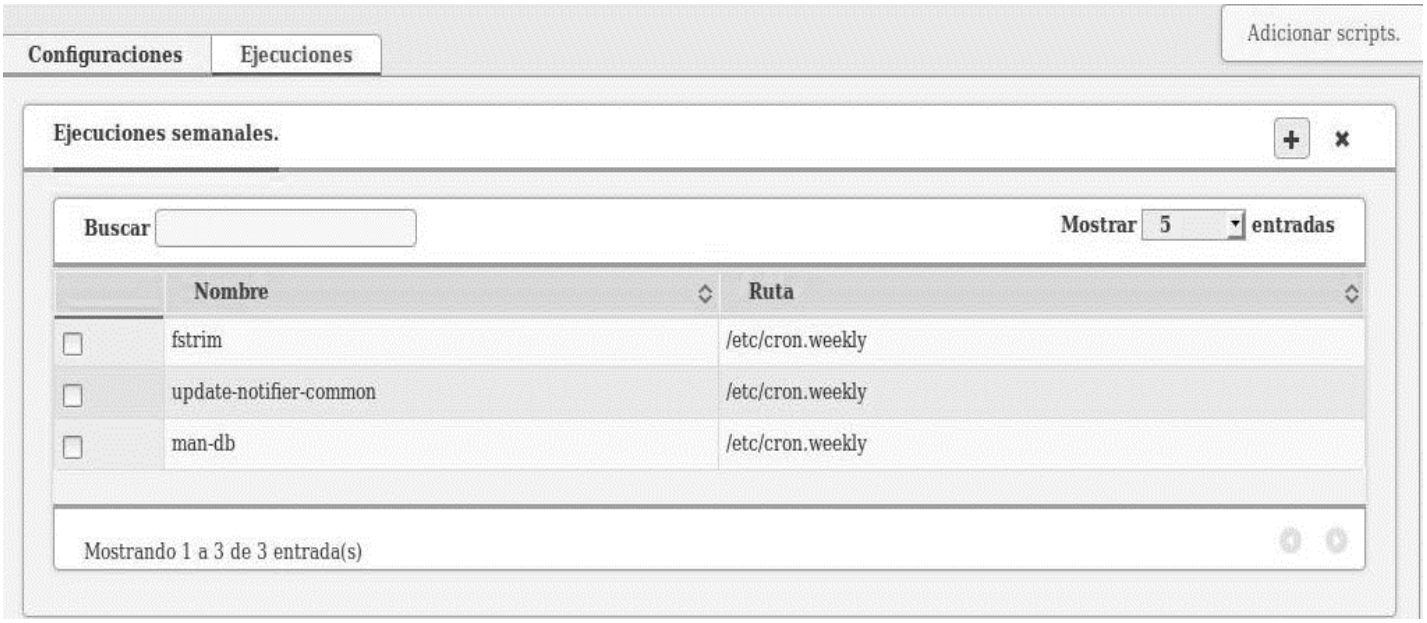
19- Adicionar ejecución realizada semanalmente.

Numero: 19	**Nombre del requisito:** Adicionar ejecución realizada semanalmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas

Descripción: Permite adicionar un script que será ejecutado semanalmente.

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones semanales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el botón de adicionar que se encuentra en la parte superior derecha y después se mostrará una interfaz que nos permite adicionar un nuevo script para que se ejecute semanalmente. Este nuevo script de añadirá al directorio */etc/cron.weekly/.* La interfaz estará compuesta por los siguientes atributos.

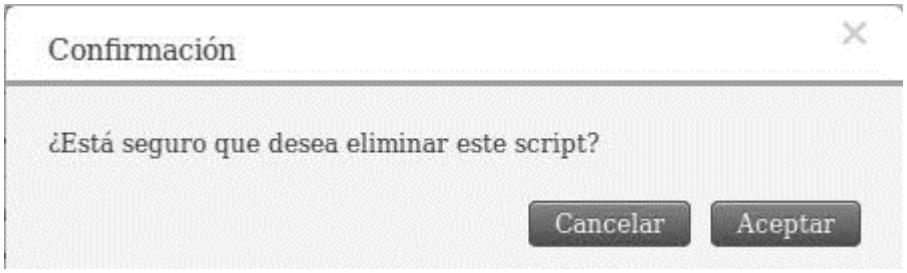
- **Nombre:** Nombre que tienes el nuevo script a adicionar. Es un campo que permite cargar el nuevo

script, del cual solo se mostrar el nombre, es un campo obligatorio.
Observaciones:
Prototipo elemental de interfaz gráfica de usuario:  La segunda interfaz de este requisito corresponde con la segunda interfaz de la historia de usuario de adicionar ejecución realizada a cada hora.

20- Eliminar ejecuciones realizadas semanalmente.

Numero: 20	**Nombre del requisito:** Eliminar ejecuciones realizadas semanalmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite eliminar scripts que se ejecuten semanalmente. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones semanales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el usuario seleccionará el o los scripts que se deseen eliminar, seleccionará el botón de eliminar que se encuentra en la parte superior derecha y después se mostrará una interfaz para que el usuario confirme la acción que va a realizar. Esta acción permite eliminar un	

script del directorio */etc/cron.weekly/.*
Observaciones:
Prototipo elemental de interfaz gráfica de usuario:



21- Editar tiempo en que se realizan las ejecuciones semanales.

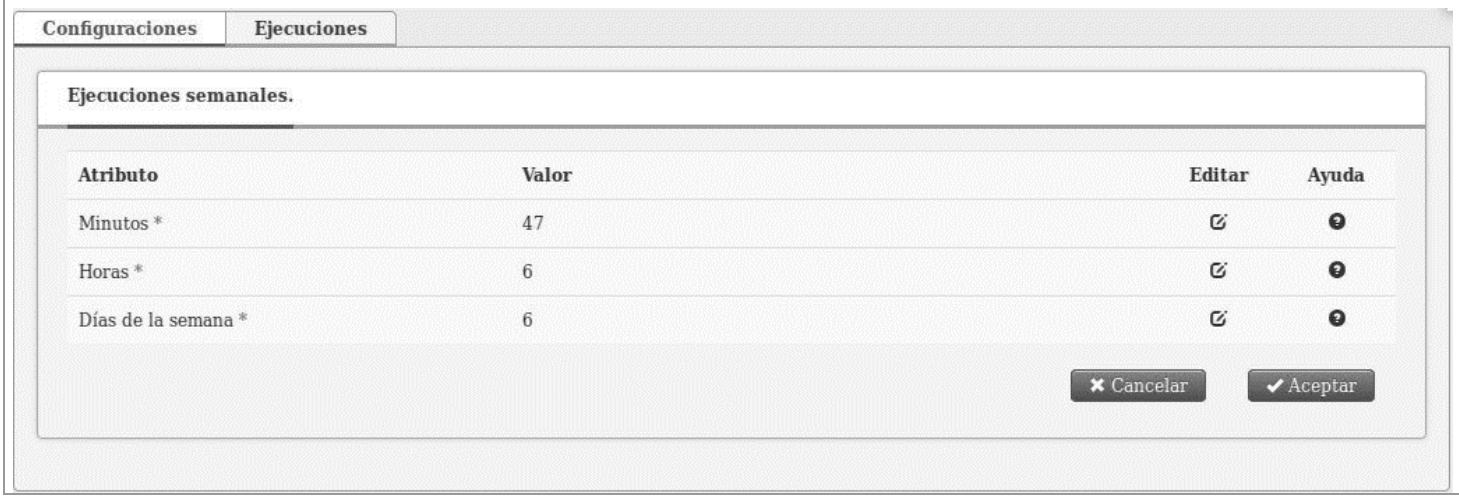
Número: 21	**Nombre del requisito:** Editar tiempo en que se realizan las ejecuciones semanales.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite editar el tiempo en que se realizan las ejecuciones semanales. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones semanales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos	

pestañas, seleccionará la pestaña *Configuraciones*. En esta se encuentran los atributos que pueden ser modificados en este tipo de ejecución con los valores que estos representan en el archivo */etc/crontab*, a continuación, se describen estos atributos.

- **Minutos:** son los minutos en los cuales se va a ejecutar la tarea. Permite caracteres como (*), (,), (/), (-) y números y tiene un rango de 0-59. Es un campo de texto y es obligatorio.
- **Hora:** es la hora en que se ejecutará la tarea, este tiene un formato de 24 horas, son números enteros y tiene un rango de 0-23. Permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.
- **Día de la semana:** es el día de la semana en el cual se va a ejecutar la tarea, presenta un rango de 0-7 siendo 0 y 7 valores correspondientes al domingo, son números enteros y permite caracteres como (*), (,), (/), (-). Es un campo de texto y es obligatorio.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



22- Listar ejecuciones realizadas mensualmente.

Número: 22	**Nombre del requisito:** Listar ejecuciones realizadas mensualmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite listar los scripts que serán ejecutados mensualmente.	

La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones mensuales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación se mostrará un listado con los nombres y la ruta de los scripts que se encuentran en el directorio */etc/cron.monthly/* los cuales son las ejecuciones realizadas mensualmente que se encuentran en el servidor.
Observaciones:
Prototipo elemental de interfaz gráfica de usuario:

23- Adicionar ejecución realizada mensualmente.

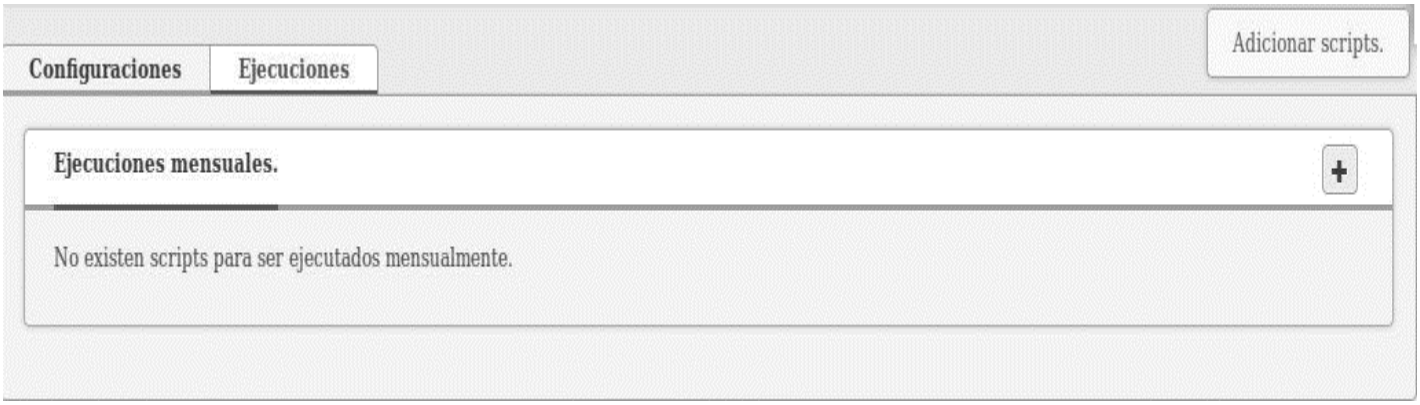
Numero: 23	**Nombre del requisito:** Adicionar ejecución realizada mensualmente.
Programador: Wilbia Esther Mariño Alcolea	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 10 horas
Riesgo en Desarrollo: Medio	**Tiempo Real:** 10 horas
Descripción: Permite adicionar un script que será ejecutado mensualmente. La funcionalidad inicia cuando el usuario selecciona la opción *Ejecuciones mensuales*, que se encuentra en el menú principal del módulo ubicado en la parte izquierda de la página. Luego se muestra una interfaz con dos pestañas, seleccionará la pestaña *Ejecuciones*, a continuación, el botón de adicionar que se encuentra en la parte superior derecha y después se mostrará una interfaz que permite adicionar un nuevo script para que se ejecute mensualmente. Este nuevo script de añadirá al directorio */etc/cron.monthly/.* La interfaz estará	

compuesta por los siguientes atributos.

- **Nombre:** Nombre que tienes el nuevo script a adicionar. Es un campo que permite cargar el nuevo script, del cual solo se mostrar el nombre, es un campo obligatorio.

Observaciones:

Prototipo elemental de interfaz gráfica de usuario:



La segunda interfaz de este requisito corresponde con la segunda interfaz de la historia de usuario de adicionar ejecución realizada a cada hora.

Anexo 2: Acta de aceptación emitida por el cliente.

Acta de aceptación de productos de trabajo

ACTA DE ACEPTACIÓN DE PRODUCTOS DE TRABAJO

En cumplimiento del **Convenio de colaboración** establecido entre el **Centro de Software Libre (CESOL)** y el estudiante **Wilbia Esther Mariño Alcolea** de la Facultad 1 de la Universidad de las Ciencias Informáticas y en función de la ejecución del proyecto: **Módulo de de planificación de tareas para HMAST**, se hace entrega del producto que se relaciona a continuación:

- **Módulo de de planificación de tareas para HMAST**

La parte Cliente, luego de haber revisado el producto de trabajo relacionado anteriormente y el cual fue desarrollado hasta un 75%, procede a firmar la aceptación de los mismos en total conformidad.

Entrega	**Recibe**
Nombre y apellidos: Wilbia Esther Mariño Alcolea	**Nombre y apellidos:** Yoandy Pérez Villazón
Cargo: Estudiante Facultad 1	**Cargo:** Director de CESOL
Firma:	**Firma:**

Universidad de las Ciencias Informáticas
Centro CESOL Facultad 1

Fecha: 1/06/2017