Universidad de las Ciencias Informáticas
Facultad 1

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Módulo para la gestión del control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST.

Autora: Ivet Campos Cesar

Tutores:

Ing. Nestor Ariel Delgado Pacheco

Ing. Yasiel Pérez Villazón

La Habana, junio 2017

¨Año 59 de la Revolución¨

Declaración de autoría

Declaro por este medio que yo Ivet Campos Cesar, con carné de identidad 90082048337 soy la autora principal del trabajo titulado “Módulo para la gestión del control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST” 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 declaración jurada de autoría en La Habana a los 30 días del mes de junio del año 2017.

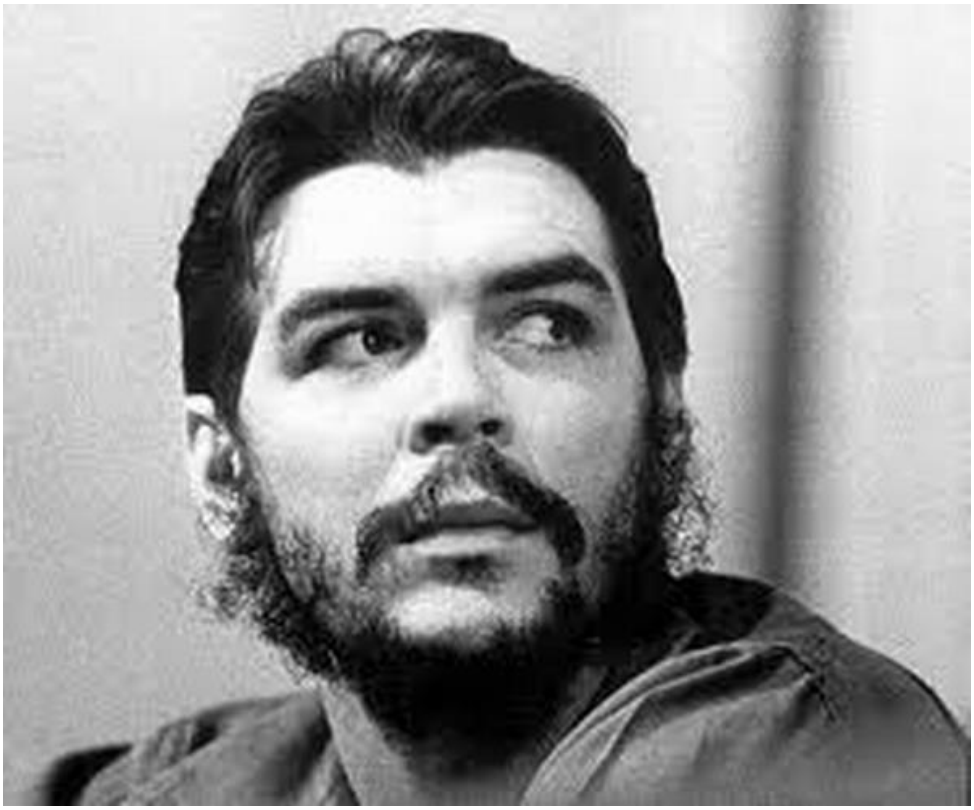
_____

Ivet Campos Cesar

_____ _____

Ing. Nestor Ariel Delgado Pacheco Ing. Yasiel Pérez Villazón

Pensamiento



"[…] si nosotros estamos hoy aquí y la Revolución Cubana está aquí, es sencillamente porque Fidel entró primero en el Moncada, porque bajó primero del Granma, porque estuvo primero en la Sierra, porque fue a Playa Girón en un tanque […] porque tiene como nadie en Cuba, la cualidad de tener todas las autoridades morales posibles para poder pedir sacrificio en nombre de la Revolución".

Ernesto Che Guevara de la Serna

Dedicatoria

- *A mis padres, que tanto dieron y sacrificaron para que me formara como la persona que soy hoy, dándome un hogar lleno de amor y cariño, defendiéndome de las personas que algún día me quisieron hacer daño, agarrándome la mano cuando era necesario que entrara en razón, y lo más importante, apostando por mí en cada cosa que hago.*
- *A mi hermano Ivancito, que tanto lucho por mi junto a mis padres cuando más nadie lo hizo y que no importa cuántos desacuerdos tengamos, para mí él siempre será mi hermano del alma.*
- *A mi novio Arian, que nunca se apartó de mí, que ha sido mi compañero inseparable estos diez últimos meses de la carrera y que con su ayuda y dedicación he llegado hasta donde estoy hoy.*
- *A mis abuelos, mis tíos y tías, mis primas y primos, en fin, toda la familia linda que tengo y que sé que estén donde estén, estarán orgullos de mí.*

Agradecimientos

- *A mis padres y mi hermano por estar ahí cada día, noche y madrugada que los necesité, dándome apoyo, cariño, meriendita y lo más importante, dándome fuerzas para seguir con mi principal objetivo en este curso que fue, graduarme.*
- *A mi novio Arian, por todos los buenos y malos momentos de la tesis y de la vida en que estuvo a mi lado, sobre todo cuando lo tenía hasta la madrugada con la luz encendida en el cuarto sin dejarlo dormir porque estaba trabajando.*
- *A mis suegros Mayra y Rafael, que me abrieron las puertas de su casa y que desde Villa Clara me brindaron su apoyo en todo momento.*
- *A mis tutores Nestor, Yasiel y Leosdanis por su ayuda incondicional en este período, por haber sido mis mejores guías y apoyo.*
- *A mis amigos y compañeros de aula por aguantarme estos cuatro años sin protestar y por ser siempre sinceros conmigo. Sobre todo, Z la bella (Zuleymis), mi compañera de desayunos y almuerzos en los días de proyecto. Gracias por ayudarme tanto.*
- *A todos mis profesores estos cinco largos años por haberme enseñado lo que hoy pongo en práctica. El esfuerzo, dedicación, conocimiento y amor que depositaron en mi enseñanza son los mejores recuerdos que tengo de esta etapa tan importante de mi vida.*
- *Al tribunal, por su ayuda y orientación este último mes.*
- *Al Fidel Castro Ruz por haberme dado la posibilidad de estar hoy aquí junto a todos ustedes.*
- *A todos los que de una forma u otra que han hecho de mí la persona que soy.*

Resumen

La Universidad de las Ciencias Informáticas forma parte del proceso de informatización de la sociedad cubana, pues desarrolla parte del *software* que desde sus inicios se usa para dicha tarea. Con el objetivo de apoyar el proceso de migración en el país está en desarrollo la Herramienta para la Migración y Administración de los Servicios Telemáticos (HMAST). Este sistema permite administrar desde plataformas libres los servicios telemáticos en las empresas cubanas tales como los servicios MySQL, DHCP[1], Apache2, Bacula y SSH[2], sin embargo, no posee un mecanismo que permita gestionar el control de acceso a recursos del sistema GNU/Linux a los usuarios de los servidores lógicos.

La presente investigación propone la realización de un módulo para HMAST, con el objetivo de gestionar desde la propia herramienta, el control de acceso a los recursos del sistema GNU/Linux por parte de los usuarios de los servidores lógicos. Se define como metodología de desarrollo AUP- Variación UCI, la cual guiará el proceso de desarrollo de *software*. Para la implementación de la solución propuesta se define como Entorno de Desarrollo Integrado (IDE) *IntelliJ IDEA*, como herramienta de modelado Visual Paradigm y como lenguaje de programación *Java* así como otros elementos de la tecnología tales como *framework spring, foreUI, git, maven, bootstrap, jquery, HTML y augeas*. Finalmente se obtuvo como resultado práctico de la investigación, un módulo integrado a HMAST que permite gestionar el control de acceso a recursos del sistema GNU/Linux a los usuarios de los servidores lógicos.

Palabras clave: acceso**,** administración, gestionar, migración**,** recursos, usuarios.

[1] *Dynamic Host Configuration Protocol, en español: protocolo de configuración dinámica de host.*
[2] ***S****ecure* ***SH****ell, en español: intérprete de órdenes seguro.*

Índice de contenido

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

Módulo para la gestión del control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST.

Introducción

La Universidad de Ciencias Informáticas (UCI), juega un papel importante en el desarrollo de la Industria Cubana del *Software*, y en la materialización de los proyectos asociados al programa cubano de informatización de la sociedad. La misma se ha convertido en una Ciudad Digital Avanzada, en la que se forma capital humano especializado, investigando y produciendo *software* para la sociedad cubana.

Como parte de dicho proceso la universidad como misión principal tiene producir aplicaciones y servicios informáticos, a partir de la vinculación estudio-trabajo como modelo de formación [1]. Dentro de su infraestructura productiva cuenta con el *Centro de Software Libre* (CESOL), líder en los procesos de migración hacia tecnologías libres y de código abierto y conformado por el Departamento de Sistemas Operativos y el Departamento de Servicios Integrales en Migración, Asesoría y Soporte (SIMAYS), este último especializado en brindar servicios relacionados con la migración a código abierto como A*sesoría, Consultoría, Capacitación y Soporte Técnico.* Este departamento, además, se encuentra proyectado hacia las siguientes líneas de trabajo: el desarrollo de estrategias, guías y planes de migración; las herramientas de soporte al proceso de migración; y la migración de los servicios telemáticos y aplicaciones de escritorio. Para dar cumplimiento a estas líneas se desarrolla la Herramienta para la Migración y Administración de Servicios Telemáticos (HMAST), solución dirigida a la administración y migración de los servicios telemáticos hacia plataformas libres de forma remota, adaptándose a las condiciones y especificidades de las empresas nacionales.

HMAST fue liberada por la Dirección de Calidad perteneciente a la UCI, registrada en el Centro Nacional de Derecho de Autor (CENDA) y actualmente en su versión 2.0. Cuenta con 5 módulos, en la v1.0 se liberó MySQL y DHCP y en la v2.0 Bácula, Apache2 y SSH. Una de las funcionalidades que posee es la gestión de servidores distribuidos por módulos que agrupan uno o varios servicios telemáticos, la cual permite crear una conexión segura con un usuario del sistema con determinados privilegios de uso sobre los recursos de este. Actualmente, HMAST no posibilita configurar los permisos a usuarios sobre los

recursos del sistema GNU/Linux, lo que dificulta realizar una mayor administración de los mismos, pues se deben poseer conocimientos avanzados del sistema GNU/Linux.

Para dar solución a la situación problemática descrita anteriormente, se presenta el siguiente **problema** de investigación: ¿Cómo configurar los permisos a los recursos del sistema GNU/Linux desde la herramienta HMAST?

A partir del problema planteado, el **objeto de estudio** de la investigación se centra en la configuración de los permisos a los recursos del sistema GNU/Linux. Enmarcándose el **campo de acción** en la configuración de los permisos a los recursos del sistema GNU/Linux desde HMAST.

Se propone como **objetivo general** desarrollar un módulo que permita gestionar el control de acceso a los recursos de los sistemas GNU/Linux por parte de los usuarios de los servidores lógicos desde HMAST.

Para cumplir el objetivo general se deben desarrollar los siguientes **objetivos específicos**:

1. Caracterizar las herramientas que posibilitan el control de acceso al uso de recursos del sistema GNU/Linux.
2. Diseñar el módulo a desarrollar de manera que se adapte a las características que posee HMAST.
3. Implementar el módulo para la gestión del control de acceso a recursos en los sistemas GNU/Linux.
4. Realizar las pruebas funcionales correspondientes al módulo implementado.

Para dar cumplimiento a los objetivos específicos se deben desarrollar las siguientes **tareas de investigación**:

1. Caracterización de los aspectos teóricos de la asignación de permisos en los sistemas GNU/Linux, para adquirir la base de la investigación.
2. Análisis de las herramientas usadas para gestionar el control de acceso a los recursos por parte de usuarios del sistema.

3. Análisis y diseño del módulo para la gestión de control de acceso en la herramienta HMAST.
4. Implementación del módulo para la gestión de control de acceso en la herramienta HMAST.
5. Diseño y ejecución de los casos de pruebas correspondientes a las funcionalidades del módulo implementado.

A partir de la investigación surgen una serie de preguntas científicas que ayudan en el proceso de desarrollo:

- ¿Qué fundamentos teóricos sustentan la gestión de control de acceso en el sistema GNU/Linux?
- ¿Cuál es el estado actual de la gestión del control de acceso a los recursos por parte de usuarios del sistema, desde herramientas existentes?
- ¿Qué elementos deben tenerse en cuenta para realizar el análisis y diseño del módulo para la gestión de control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST?
- ¿Qué métodos o técnicas aplicar para la codificación y posterior validación del módulo para la gestión de control de acceso en la herramienta HMAST?

Para dar cumplimiento al objetivo propuesto y brindar solución a las tareas de investigación, se utilizaron los siguientes **métodos de la investigación científica**:

Inductivo-deductivo: Permite llegar a proposiciones generales a partir de los hechos que conforman la teoría, o a partir de dichas teorías arribar a conclusiones sobre casos particulares que se llevaron a la práctica. En la investigación es usado este método para, a partir del análisis de las herramientas existentes que realizan la gestión de control de acceso a recursos en los sistemas GNU/Linux, definir las características que va a poseer el módulo para la herramienta HMAST.

Modelación: Es utilizado para confeccionar los diagramas correspondientes a la representación de la propuesta de solución.

Introducción

Módulo para la gestión del control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST.

Los **métodos empíricos** utilizados en el proceso investigativo fueron el **análisis documental** y la **tormenta de ideas**. El **análisis documental** es empleado para la revisión de la literatura necesaria durante la investigación sobre el proceso de asignación o revocación de permisos en los sistemas GNU/Linux. El método **tormenta de ideas** es utilizado para el levantamiento de requisitos, así como para la determinación de las restricciones de *software* e implementación que deben tenerse en cuenta en el proceso de desarrollo.

El presente documento cuenta con introducción, desarrollo y conclusiones, además de recomendaciones, referencias bibliográficas y anexos. En el desarrollo se encuentran 3 capítulos que se describen a continuación:

Capítulo 1: Fundamentación teórica: Se abordan conceptos claves que serán usados durante el desarrollo de la investigación y se describe la herramienta HMAST, se fundamentan las tecnologías, lenguajes de programación, metodología y herramientas utilizadas en el desarrollo del sistema.

Capítulo 2: Diseño del sistema. Se especifican las características que tendrá la propuesta de solución que se propone y se exponen las funcionalidades del módulo propuesto. También se realiza la descripción de las Historias de Usuario donde se presentan los prototipos de interfaz de cada una de las funcionalidades y se explica la arquitectura de la herramienta.

Capítulo 3: Implementación y pruebas. Se implementa la propuesta de solución. Se explican las clases principales del sistema, se confecciona el plan de pruebas y se realizan las pruebas necesarias para validar que las funcionalidades desarrolladas dan cumplimiento a los requisitos planteados.

Capítulo 1. Fundamentación Teórica

En el presente capítulo se realiza un estudio de las diferentes herramientas que se utilizan en los sistemas GNU/Linux para gestionar el control de acceso a los ficheros y directorios del sistema, así como de los conceptos relacionados. Se detallan las herramientas y tecnologías que serán empleadas para el desarrollo del módulo en cuestión, como son el lenguaje de programación y la metodología de desarrollo de *software*. También se caracteriza HMAST, así como su arquitectura y se exponen algunas consideraciones a tener en cuenta para integrar el módulo a esta herramienta.

1.1 Control de acceso

El proceso de administración de identidades y de privilegios de los usuarios en los sistemas de información permite reducir el riesgo de acceso o modificación no autorizada a esta. El mantener el acceso a los sistemas, a los usuarios válidos con los privilegios correctos requiere la definición de políticas de control de acceso, este último consiste en el proceso de conceder permisos a usuarios o grupos para el acceso a datos o recursos [2].

Los controles de acceso se agrupan en físicos (un edificio, un local), lógicos (un sistema operativo o una aplicación informática específica) y administrativos. Una definición más sintetizada de control de acceso abarcaría únicamente la aprobación de acceso, por lo que el sistema adopta la decisión de conceder o rechazar una solicitud de acceso de un sujeto ya autenticado sobre la base a lo que el sujeto está autorizado a acceder. Esto hace que el control de acceso se convierta en un elemento necesario para proteger la confidencialidad, integridad y disponibilidad de los objetos, pues permite que los usuarios autorizados accedan solo a los recursos que se requiera para realizar sus tareas [2]. Sobre el acceso a los recursos, los métodos para asignar permisos en los sistemas GNU/Linux y demás aspectos importantes tratan los epígrafes que se exponen a lo largo del capítulo.

1.2 Usuarios y grupos en el sistema GNU/Linux

Los sistemas GNU/Linux son sistemas multiusuarios y la tarea de añadir, modificar, eliminar y en general

administrar usuarios y grupos, es un elemento de seguridad que mal administrado puede convertirse en un problema de seguridad. Para ello en estos sistemas los usuarios se identifican por un número único de usuario (*User Identifier*, UID[3]) y pertenecen a un grupo principal de usuarios, aunque pueden pertenecer a más grupos además del principal, estos últimos se identifican también por un número único de grupo (Group ID, GID) [3].

Existe en GNU/Linux una serie de usuarios predefinidos, de los cuales los más típicos son los que se describen a continuación [3]:

- **Usuario root**

También llamado superusuario o administrador y su UID es 0 (cero). Es la única cuenta de usuario con privilegios sobre todo el sistema, cuenta con acceso total a todos los archivos y directorios con independencia de propietarios y permisos. Controla la administración de cuentas de usuarios, ejecuta tareas de mantenimiento del sistema y puede detenerlo. Instala *software*, además de modificar o reconfigurar el núcleo o kernel, etc.

- **Usuarios especiales**

Los usuarios presentes en esta categoría son: ***bin***, ***daemon***, ***adm***, ***lp***, ***sync***, ***shutdown***, ***mail***, ***operator***, ***squid***, ***apache***.

Son llamados también cuentas del sistema. No tienen todos los privilegios del usuario root, pero dependiendo de la cuenta, asumen distintos privilegios de root, esto para proteger al sistema de posibles formas de vulnerar la seguridad. No tienen contraseñas pues son cuentas que no están diseñadas para iniciar sesiones con ellas por lo que se les conoce como cuentas de "no inicio de sesión" (*nologin*). Se crean automáticamente al momento de la instalación del sistema. Generalmente se les asigna un UID entre 1 y 100 (definido en */etc/login.defs*)**.**

- **Usuarios estándar**

[3] *UID: Identificador de Usuario, que para todo es único*

Se usan para usuarios individuales y cada usuario dispone de un entorno o directorio de trabajo, ubicado generalmente en la carpeta /home/nombre-usuario que puede personalizarlo según sus necesidades. Tienen solo privilegios completos en su directorio de trabajo y se les asigna generalmente un UID superior a 500.

El acceso de los usuarios y grupos de usuarios a ciertos recursos del sistema depende de los permisos que poseen para ello, esto con el fin de permitir o no, leer, modificar y/o ejecutar archivos y directorios, logrando administrar y a la vez proteger la integridad del sistema.

1.3 Tipos de permisos en los sistemas GNU/Linux.

Los permisos se asocian o asignan a usuarios y grupos específicos sobre archivos y directorios, estos permisos vistos de manera básica son [4]:

Lectura: del inglés ***read*** y abreviado como ***r***, que indica que se puede leer el archivo, y además listarlo con el comando básico ***ls***.

Escritura: del inglés ***write*** y abreviado como ***w***, que una vez activo permite al usuario realizar modificaciones en el archivo.

Ejecución: del inglés ***execute*** y abreviado como ***x***, permite al usuario la ejecución del archivo.

Estos permisos están organizados en una triada binaria, tienen un valor específico y pueden tener dos estados, si el bit está en 1 quiere decir que está activo, por el contrario, si el bit está en 0, está inactivo. En la figura 1 se muestra como está estructurada dicha representación numérica:

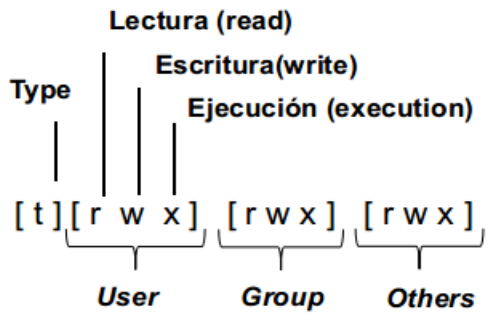


Figura 1. Representación Numérica de los permisos en GNU/Linux.

Cada categoría de permisos se representa con tres caracteres claramente diferenciados. El primer conjunto representa la categoría Permisos del Usuario o propietario (del inglés ***user***, abreviada como ***u***), el segundo conjunto representa la de Permisos del Grupo (del inglés ***group***, abreviada como ***g***) y el tercer conjunto representa los Permisos del resto de usuarios (o también llamados "los otros" del inglés ***others***, abreviada como ***o***)). Cada uno de los caracteres contiene los permisos de lectura, escritura y ejecución respectivamente y son definidos a continuación [4].

Permisos del usuario o propietario

El propietario es aquel usuario que genera o crea un archivo/carpeta dentro de su directorio de trabajo (*HOME*), o en algún otro directorio sobre el que tenga derechos. Cada usuario puede crear los archivos que requiera dentro de su directorio de trabajo y los únicos usuarios con derechos sobre estos archivos y directorios serán en principio el usuario *root* o administrador y el propio usuario [4].

Permisos del grupo

Generalmente cada usuario pertenece a un grupo de trabajo, de esta forma, cuando se gestiona un grupo, se gestionan todos los usuarios que pertenecen a este. Por tal razón es más fácil integrar varios usuarios en un grupo al que se le conceden determinados privilegios en el sistema, que asignar los privilegios de

forma independiente a cada usuario [4].

Permisos del resto de usuarios

Son aquellos permisos que se les otorgan a los usuarios que no pertenecen al grupo de trabajo en el que se encuentran los archivos sobre los cuales se gestionan los permisos, pero que pertenecen a otros grupos de trabajo. Estos son llamados “otros” o “resto de usuarios”. Es decir, los privilegios de los archivos contenidos en cualquier directorio, pueden tenerlos usuarios que no pertenezcan al grupo de trabajo en el que está integrado el archivo en cuestión [4].

1.4 Herramientas para la administración de los permisos

Actualmente existen varias herramientas que son utilizadas para asignar privilegios a usuarios y grupos sobre archivos y directorios, así como para ejecutar determinadas tareas sobre el sistema. A continuación, se describen estas herramientas, detallándose las principales funcionalidades y características de cada una.

1.4.1 *Chmod*

Chmod ("change mode", del inglés *cambiar modo*) es una llamada al sistema y su comando asociado en los sistemas operativos GNU/Linux (estandarizados en POSIX[4] y otros estándares) es de igual nombre *chmod* y permite cambiar los permisos de acceso de un fichero o directorio, lo que quiere decir que sólo el propietario puede cambiar los permisos del archivo (además, del administrador del sistema) [5].

Chmod utiliza las estructuras en modo octal y en modo simbólico:

- **Modo octal**

La notación octal consiste en valores de tres a cuatro dígitos en base 8. Con esta notación cada número representa un componente diferente de permisos a establecer: clase de usuario, clase de grupo y clase de

[4] *Interfaz de Sistema Operativo Portable, del inglés Portable Operating System Interface.*

otros, respectivamente. Cada uno de estos dígitos es la suma de los bits que lo componen (en el sistema numeral binario). Como resultado, bits específicos se añaden a la suma, y son representados por un numeral [5]:

- El bit de ***ejecución*** (acceso en el caso de directorios) añade 1 a la suma.
- El bit de ***escritura*** añade 2 a la suma
- El bit de ***lectura*** añade 4 a la suma.

En este caso la estructura para asignar los permisos es: [5]

chmod [NML] [nombre_archivo]

Donde:

[N] = hace referencia al usuario propietario y es un valor comprendido entre 0 y 7.

[M] = hace referencia al grupo de usuarios y es un valor comprendido entre 0 y 7.

[L] = hace referencia al resto de los usuarios y es un valor comprendido entre 0 y 7.

A su vez, el número indica el tipo de permiso que tiene cada usuario, por lo que un número igual a 0 (cero) indica que no tiene ningún permiso y un valor igual a 7 (siete) indica que tiene todos los permisos. El valor dentro de ese rango, indicará el permiso para cada tipo de usuario como se muestra a continuación:

[NML] = [0] = No tiene ningún permiso.

[NML] = [1] = Solo permite la ejecución.

[NML] = [2] = Solo permite la escritura.

[NML] = [3] = Permite la escritura y ejecución.

[NML] = [4] = Solo permite la lectura.

[NML] = [5] = Permite la lectura y ejecución.

[NML] = [6] = Permite la lectura y escritura.

[NML] = [7] = Permite lectura, escritura y ejecución.

Por tanto, y para los tres tipos de usuarios (propietario, grupo y todos los demás) el grado de permiso dependerá del valor que se le asigne con el comando *chmod*.

Ejemplo:

```
ivet@ivet-System-Product-Name: ~$ chmod 777 Caso.odt
```

Con esta orden se está indicando que todos los usuarios (propietario, grupo y otros) tienen permisos para leer, escribir y ejecutar el archivo *Caso.odt* debido a que el primer valor octal (siete) indica que el usuario propietario tiene permitida la lectura, escritura y ejecución, el segundo valor octal (siete) indica que el grupo propietario tiene permitida la lectura, escritura y ejecución, y el ultimo valor octal (siete) indica que el resto de usuarios tiene permitida la lectura, escritura y ejecución.

- **Modo simbólico**

Permite establecer los permisos de un archivo o directorio a través de identificadores del bit (**r, w,** o **x**) de los permisos. En este método, solo hay que tomar en cuenta que, partiendo de los permisos ya establecidos, se quitan o se agregan a los ya existentes. Su estructura para ejecutar la asignación o revocación es la siguiente [5]:

chmod [usuario] [opción][permiso] [nombre_archivo]

Donde:

[usuario] = usuario al que se les otorga o se le quita el permiso.

[opción] = hace referencia a si se permite o deniega el permiso a un usuario o grupo de usuarios.

[permiso] = permiso de lectura, escritura y ejecución.

[nombre_archivo] = hace referencia al archivo, carpeta o partición a la que afecta el permiso.

- Según el tipo de **[usuario]** al que le afecta el permiso, se tiene que: [5]

[u] = propietario del archivo o user

[g] = grupo de usuarios o group

[o] = otros usuarios u others

[a] = todos los usuarios o ***all***

- En cuanto a las **[opciones]** tiene dos variantes: [5]

  [+] = otorgar permiso **[-]** = denegar permiso

- Según el tipo de **[permiso]**, las posibilidades son tres para cada usuario: [5]

  [r] = permiso de ***lectura*** **[w]** = permiso de ***escritura*** **[x]** = permiso de ***ejecución***

Ejemplos:

1. Asignar permisos de lectura, escritura y ejecución para los usuarios "otros" a todos los archivos de la carpeta *Descargas.*

```
ivet@ivet-System-Product-Name:~$ chmod o=rwx *
```

2. Asignar todos los permisos a todos los usuarios para el archivo fichero.txt

```
ivet@ivet-System-Product-Name:~$ chmod a=rwx fichero.txt
```

1.4.2 Listas de Control de Acceso.

Las listas de control de acceso (**ACL**, ***a**ccess **c**ontrol **l**ists*) son usadas para controlar los permisos de acceso de los archivos y directorios. Cada objeto del sistema puede ser asociado a una **ACL** que controla el acceso de modo discrecional hacia ese objeto. Estas proveen un mecanismo para especificar permisos

en archivos y otros objetos de forma mucho más flexible que el sistema estándar Unix de usuario/grupo/otros [6].

Las listas de control se almacenan en el mismo sistema de archivos, por ello existen parches disponibles para los sistemas de archivos ext2 y ext3 que habilitan el soporte de estas mediante atributos extendidos (EAs).

Existen dos tipos de ACLs: *por defecto* y *predeterminada*. Por defecto es la lista de control de acceso para un archivo o directorio específico y predeterminada significa que solo puede ser asociada con un directorio. Ejemplo si un archivo dentro de un directorio no tiene una *acl* por defecto, utiliza las reglas de la *acl* predeterminada para el directorio [7].

Estas se pueden configurar por usuario, grupo, a través de la máscara de derechos efectivos y para el resto de usuarios. [7]

Su sintaxis contiene tres campos separados por "**:**". [8]

[Tipo]: [Calificador]: ListaPermisos [...]

Tipo:

- "***u***" Usuario
- "***g***" Grupo
- "***o***" Otros
- "***m***" Máscara

Calificador:

- UID (ID de usuario)
- GID (ID de grupo)
- Vacío (Asume UID, GID del creador)

Lista de Permisos:

El tercer campo es el de acceso y puede ser representado de 2 maneras:

- Cadena estándar **rwx** (las cadenas se pueden remplazar por "-" si no se requiere dar acceso de ese tipo)
- Cadena simbólica: **+ ^**.

Esta sintaxis se utiliza en el proceso de configuración acompañada de las órdenes: *getfacl* y *setfacl* que permiten obtener y modificar los permisos sobre ficheros y directorios de manera específica [6].

- **Getfacl**

El comando ***getfacl*** (get file access control list) se utiliza para determinar los permisos establecidos en las listas de control de acceso de un archivo o directorio dado. Su sintaxis básica es [6]:

getfacl [opciones] ....

Las opciones más utilizadas son [6]:

- **-R** ***//***Cambia permisos a archivos y directorio de forma descendente a partir de un directorio dado.
- **-d** //Muestra los permisos por defecto.

El listado de permisos que se obtiene al ejecutar *getfacl* se compone de entradas de tipo *user* y *group*, además de las entradas para *mask* y *other*. En el caso de las entradas *user* y *group*, siempre se tendrá una entrada para el propietario del fichero y el grupo del fichero (son las líneas que no indican un usuario o grupo concreto) y tantas líneas adicionales como Entradas de Control de Acceso (ACEs) de ese tipo se hayan asignado al fichero o directorio. La entrada *other* es la entrada del permiso tradicional otros del modelo UGO (Usuario, Grupo y Otros). [6]

Ejemplo de la ejecución del comando *getfacl*:

```
ivet@ivet-System-Product-Name: ~$ getfacl /home/ivet/Escritorio/caso.odt
# file: caso.odt
# owner: ivet
# group: root
user::rwx
group::rwx
other::r-x
```

- Con esta sentencia en la consola se obtienen los permisos que tienen los usuarios, grupos y otros

sobre el fichero *caso.odt.*

- **Setfacl**

El comando ***setfacl*** (set file access control list) se utiliza principalmente para [6]:

- Asignar permisos básicos a: Usuario, Grupo, Otros (UGO). Equivalente a lo que hace el comando: *chmod*.
- Asignar permisos por defecto (Para nuevos directorios/archivos).
- Asignar permisos máximos (Máscaras).
- Asignar permisos adicionales a grupos y usuarios.

Las opciones que se usan del comando setfacl son [6]:

- **-m –modify** //Indica que se modificará el ACL un directorio.
- **-x --remove** //Indica que se eliminará el ACL de un directorio.
- **-b --remove-all** //Indica que se eliminarán todos los ACL de un archivo o directorio.
- **-k --remove-default** //Indica que se eliminarán todos los ACL predeterminado de un archivo o directorio.
- **--mask** //Fuerza el cálculo de la máscara de los permisos efectivos de un ACL, aún si se ha proporcionado una de manera explícita.
- **--d –default** //Todas las operaciones se aplican solo en el ACL predeterminado.
- **-R** //Aplica los cambios de ACL de modo descendente en un directorio.

- **Ejemplo del uso del comando *setfacl*:**

Se desea otorgar permisos de lectura y ejecución al grupo "www-data" (Apache). Primero para listar la ACL se usa el comando antes estudiado *getfacl* obteniendo los siguientes resultados:

```
ivet@ivet-System-Product-Name: ~$ getfacl /var/www/
getfacl: Removing leading '/' from absolute path names
# file: var/www/
# owner: root
# group: root
user::rwx
group::rwx
other::---
```

Estos valores por defecto en la *acl* se actualizan automáticamente cada vez que se modifican los permisos en el sistema de archivos. Una vez listado los valores, para dar lectura y ejecución al grupo "www-data" se utiliza el comando *setfacl* para modificar los permisos ya existentes, de la siguiente forma:

```
ivet@ivet-System-Product-Name: ~$ setfacl -m group:www-data:r-x /var/www/
```

La opción **"-m"** indica que se está modificando la *acl*. A continuación, se indican los permisos utilizando la sintaxis "group: [nombre-grupo]: [permisos]" y finalmente se indica la ruta del directorio. Una vez modificada la *acl* se verifica nuevamente con "*getfacl*":

```
ivet@ivet-System-Product-Name: ~$ # getfacl /var/www
# file: var/www/
# owner: root
# group: root
user::rwx
group::rwx
group:www-data:r-x
mask::rwx
other::---
```

1.4.3 Sistema de permisos mediante Bits SUID.

El **bit SUID** o ***setuid*** (del inglés Set User ID) es una extensión del permiso de ejecución. Se utiliza para que usuarios ajenos al directorio *home* de un usuario determinado, al ejecutar una orden, ésta se ejecute con permisos del usuario propietario en lugar de hacerlo con los del usuario que ejecuta la orden, es decir,

es equivalente a que sea ejecutada por el propietario [6].

Para activar el bit SUID, se puede ejecutar de dos maneras, la primera utilizando el comando *chmod* en su modo simbólico con la sintaxis: *chmod u+s nombre_archivo* donde una vez ejecutado en consola, en el resultado que se muestra de cómo queda el bit activado en vez de la '**x**' en el grupo del usuario o propietario aparecería una '**s**' de *suid*. Ejemplo:

```
ivet@ivet-System-Product-Name: ~$ chmod u+s /home/ivet/Escritorio/A/
ivet@ivet-System-Product-Name: ~$ ls -ld /home/ivet/Escritorio/A/
drws------ 3 ivet ivet 4096 may 30 09:09 /home/ivet/Escritorio/A/
```

La otra opción sería utilizando el mismo comando *chmod,* pero en su modo octal y se sumaría 4000 al número que representa el permiso que tiene [6].

1.4.4 Sistema de permisos mediante Bits SGID.

El **bit SGID** o ***setgid*** (del inglés Set Group ID), de forma análoga al caso de *setuid*, lo que determina es que el proceso explicado anteriormente se pueda ejecutar con los permisos del grupo al que pertenece el archivo. Si se aplica el **bit SGID** a una carpeta, todas las subcarpetas y archivos creados dentro de dicha carpeta tendrán como propietario el grupo propietario de la carpeta en lugar del grupo primario del usuario que ha creado el archivo, por otra parte, si se otorga permisos de lectura y escritura al grupo, los archivos podrán ser modificados por todos los miembros del grupo y en el caso de que cualquiera de estos usuarios cree un archivo, éste pertenecerá al grupo [6].

Ejemplo: Si el usuario pertenece al grupo 'ventas' y existe un binario llamado 'reporte' que su grupo es 'ventas' y tiene el bit SGID activado, entonces el usuario que pertenezca al grupo 'ventas' podrá ejecutarlo.

1.4.5 Sistema de permisos mediante Bit de persistencia (Sticky Bit).

El bit de persistencia o ***sticky bit*** significa que un usuario sólo podrá modificar y eliminar archivos y directorios subordinados dentro de un directorio que le pertenezca. En ausencia de éste, se aplican las reglas generales y el derecho de acceso de escritura sólo permite al usuario crear, modificar y eliminar

archivos y directorios subordinados dentro de un directorio. Los directorios a los cuales se les ha establecido ***sticky bit*** restringen las modificaciones de los usuarios a sólo adjuntar contenido, manteniendo control total sobre sus propios archivos y permitiendo crear nuevos archivos; este bit de persistencia sólo permitirá adjuntar o añadir contenido a los archivos de otros usuarios [6]. Es utilizado en directorios como ***/tmp*** y ***/var/spool/mail*** y se indica con una '**t**' como el ejemplo siguiente [6]:

```
ivet@ivet-System-Product-Name: ~$ ls -ld /tmp
drwxrwxrwt 6 root root 4096 nov 30 8:14 /tmp
```

Puede apreciarse la '**t**' en vez de la '**x**' en los permisos de los usuarios ***others***. Lo que hace el bit de persistencia en directorios compartidos por varios usuarios es que sólo el propietario del archivo pueda eliminarlo del directorio. Es decir, cualquier otro usuario va a poder leer el contenido de un archivo o ejecutarlo si fuera un binario, pero sólo el propietario original podrá eliminarlo o modificarlo. Si no se tuviera el sticky bit activado, entonces en estas carpetas públicas, cualquier usuario podría eliminar o modificar los archivos de cualquier otro usuario.

Para cambiar este tipo de bit se utiliza el comando ***chmod,*** agregando un número octal (1 al 7) extra al principio de los permisos, ejemplo:

```
ivet@ivet-System-Product-Name:~$ ls -l /usr/local/nuevo
total 32
drwxr-xr-x 2 root root 4096 nov 30 08:16 nuevo
# Uso del comando chmod
ivet@ivet-System-Product-Name:~$ chmod 4511 /usr/local/nuevo
ivet@ivet-System-Product-Name:~$ ls -l /usr/local/nuevo
dr-s--x--x 2 root root 4096 nov 30 08:18 nuevo
```

En este caso el valor extra es el número cuatro y los demás permisos mantiene su valor. Es decir, los permisos originales en este ejemplo eran 511 (rwxr-xr-x), y al cambiarlos a 4511, se cambió el bit SUID reemplazando el bit 'x' del usuario por 's'.

1.4.6 Sistema de permisos preestablecidos con umask.

La máscara de usuario (*umask*, abreviatura de ***u**ser **mask***) es una función que establece los permisos predeterminados para los nuevos archivos y directorios creados en el sistema. Puede establecerse en notación octal de tres o cuatro dígitos o bien en notación simbólica. Puede establecerse cualquier valor para *umask*, pero debe tomarse en consideración que esta jamás permitirá crear nuevos archivos ejecutables. Cada usuario tiene su máscara por defecto y esta se puede fijar para todos los usuarios en el archivo /etc/profile o para cada usuario en el archivo /home/usuario/. bashrc [6].

Cuando se utiliza la notación octal de cuatro dígitos, el primer dígito siempre corresponde a los permisos especiales, pero el valor de éste siempre será 0; el segundo dígito corresponde a la máscara de la clase otros; el tercer dígito corresponde a la máscara para la clase de grupo; y el cuarto dígito corresponde a la máscara para la clase de usuario [6].

```
.----------- Permisos especiales (siempre es 0 en umask)
| .---------- Clase de otros
| | .-------- Clase de grupo
| | | .------ Clase de usuario
| | | |
↓ ↓ ↓ ↓
0 0 2 2
```

Para establecer el valor de la máscara, se usa el mismo comando *umask* seguido del valor de máscara que se desee, este corresponde a los bits contrarios del permiso predeterminado que se quiera asignar.

Ejemplo de uso de *umask*:

```
ivet@ivet-System-Product-Name:~$ umask
0002
ivet@ivet-System-Product-Name:~$ mkdir carpeta
ivet@ivet-System-Product-Name:~$ ls -l
total 40
drwxrwxr-x 2 ivet ivet 4096 mar 23 13:34 carpeta
drwxr-xr-x 2 ivet ivet 4096 mar 23 11:37 Descargas
```

```
ivet@ivet-System-Product-Name:~$ umask 022
ivet@ivet-System-Product-Name:~$ mkdir carpeta1
ivet@ivet-System-Product-Name:~$ ls -l
total 44
drwxrwxr-x 2 ivet ivet 4096 mar 23 13:34 carpeta
drwxr-xr-x 2 ivet ivet 4096 mar 23 13:35 carpeta1
drwxr-xr-x 2 ivet ivet 4096 mar 23 11:37 Descargas
ivet@ivet-System-Product-Name:~$ umask
0002
```

La modificación con umask de la máscara por defecto no afecta a los archivos y carpetas existentes sino sólo a los nuevos que cree ese usuario a partir de ese momento.

1.4.7 Comando *su* y *sudo*.

Además del acceso que puedan tener los usuarios, los administradores pueden asignar derechos específicos a las cuentas de grupo o a cuentas de usuario individuales. Estos derechos autorizan a los usuarios a realizar acciones específicas tales como iniciar una sesión en un sistema de forma interactiva, ejecutar comandos como si fueran administradores o realizar copias de seguridad de archivos y directorios, entre otras operaciones que son posibles mediante la asignación de privilegios usando los comandos *su* y *sudo*.

1.4.7.1 Comando *su*.

Una de las formas de elevar los privilegios de un usuario estándar es usando el comando ***su*** (*switch user o "cambiar de usuario"*). Este comando ejecuta una nueva sesión en la terminal o consola como otro usuario, en general, el usuario *root*, de esta manera, el usuario estándar en cuestión se convierte en el usuario administrador hasta cerrar esa sesión [9]. Se debe tener en cuenta que para usar el comando *su* el usuario debe conocer la contraseña del usuario *root*, por lo que una vez conocida tendrá acceso total al sistema.

La sintaxis de este comando es la siguiente [9]:

su – nombre de usuario

Ejemplo:

```
ivet@ivet-System-Product-Name: ~$ su – ivet
```

Una vez ejecutado el comando se le pedirá la contraseña del usuario que ejecuta el comando. El guión (-) permite el inicio de un nuevo shell de conexión con las preferencias del usuario *ivet*. Si se omite el guión no se cargará la sesión desde el directorio *home* del usuario *ivet* y no se inicializarán las variables de preferencia para ese usuario (*HOME, SHELL, USER, LOGNAME y PATH*). Si ejecuta el comando ***su*** o ***su -*** por sí solo, sin especificar ningún usuario, se da por sentado que está invocando al usuario root [9].

1.4.7.2 Comando *sudo*.

Sudo (del inglés ***s****uper* ***u****ser* ***do***[5][6]) es una herramienta de los sistemas operativos tipo Unix, como GNU/Linux, o Mac OS X, que permite que un administrador del sistema delegue autoridad para dar a ciertos usuarios (o grupos de usuarios) la capacidad de ejecutar algunas órdenes (o la totalidad) como root u otro usuario, al tiempo que permite auditar el rastro de las órdenes dadas y sus argumentos [10].

Por defecto, el usuario debe autenticarse con su contraseña al ejecutar *sudo*. Una vez se ha autenticado el usuario, y si el archivo de configuración /etc/sudoers permite dar al usuario acceso al comando requerido, el sistema lo ejecuta. En el caso de que un usuario necesite ejecutar un comando que requiera privilegios elevados u otros comandos, deberá anteponer la palabra *sudo* al comando que se desea ejecutar de la forma: ***sudo comando_root*** y para lo cual deberá realizar el proceso de autenticación antes explicado [10].

Este archivo de configuración /etc/sudoers especifica qué usuarios pueden ejecutar qué comandos en nombre de qué otros usuarios. Se encarga de determinar los privilegios que tiene cada usuario para

[5] S*u: Cambiando quien dices ser* (*Practical Unix & Internet Security*. 2nd Ed, Capítulo 4.3). (en inglés)

[6] *Simson Garfinkel, Gene Spafford. Practical Unix & Internet Security. 2nd Ed. 1996. O'Reilly & Associates, Inc., p. 84.*

ejecutar *sudo* y de configurar todos los directorios incluidos mediante sentencias *includedir*, y opcionalmente soporta LDAP[7] [10]. Para su edición existe la utilidad *visudo*, que invoca al editor que se tenga por defecto que generalmente es ***vi***. Cuando *visudo* es usado, bloquea el archivo /etc/sudoers de tal manera que ningún otro usuario lo puede utilizar, esto por razones de seguridad que evitarán que dos o más usuarios administradores modifiquen accidentalmente los cambios que el otro realizó. Además, verifica que el archivo esté bien configurado, es decir, detectará si hay errores de sintaxis principalmente en sus múltiples opciones o reglas de acceso que se tengan [10].

Está compuesto por dos tipos de entradas: alias (variables) y especificaciones de usuario (que definen quién puede ejecutar qué). Un alias se refiere a un usuario, un comando o a un equipo y dependiendo del tipo de alias serán sus elementos. Existen cuatro tipos [10]:

- User_Alias
- Runas_Alias
- Host_Alias
- Cmnd_Alias.

Cada definición de alias es de la forma: NOMBRE_DEL_ALIAS TIPO_DE_ALIAS = valor1, valor2, ..., etc. Donde los nombres de alias deben comenzar con una letra mayúscula. [10]

Para los de usuario (User_Alias) se debe especificar una lista de usuarios que soporta el uso de nombres de usuario, ids de usuario (prefijados con un carácter '#'), nombres de grupo e ids de grupo (prefijados con '%' y '%#' respectivamente), grupos de red (prefijados con '+') y otros alias. Es posible utilizar comillas dobles para evitar caracteres especiales y espacios. [10] Los alias para "ejecutar como" (Runas_Alias) se especifica igual que los de usuario. Para especificar los de host (Host_Alias) se pueden utilizar nombres de host, direcciones IP, direcciones de red, grupos de red y otros alias de host.

Los alias de comando (Cmnd_Alias), es posible especificar uno o más nombres de comandos, directorios y otros alias de comando. Un nombre de comando es un nombre de archivo calificado (con la ruta absoluta al binario/ejecutable) que puede incluir comodines (*wildcards*). Un nombre de archivo simple permite a un usuario ejecutar el comando con cualquier argumento que desee. Sin embargo, es posible

[7] *Protocolo Ligero de Acceso a Directorio, del inglés Lightweight Directory Access Protocol.*

especificar los argumentos de línea de comandos que se permiten (incluyendo comodines), para que el usuario ejecute únicamente lo que necesita (principio de mínimo privilegio). De forma alternativa, es posible especificar **" "** para indicar que el comando sólo puede ser ejecutado sin argumentos. Pero si un comando tiene argumentos asociados, entonces deben coincidir exactamente con aquellos enviados por el usuario en la línea de comandos. [10]

Las especificaciones de usuario determinan qué comandos un usuario puede ejecutar (y como qué usuario) en los hosts especificados. Por defecto los comandos se ejecutan como root, pero este comportamiento se puede cambiar en cada comando. [10]

- **Ejemplo:**

Para permitir que un usuario estándar, en este caso el usuario ***ivet*** pueda ejecutar cualquier comando como administrador (e incluso abrir una sesión como administrador ejecutando *sudo su*) se escribe la siguiente línea en el archivo /etc/sudoers: **ivet ALL=(ALL) ALL**

Considerando que ALL significa "todos", esto quiere decir que la regla permite que el usuario ***ivet***, en todos los hosts, puede ejecutar como todos los usuarios todos los comandos. Esta es una configuración muy básica, sin embargo, es posible configurar *sudo* para que un usuario estándar pueda ejecutar sólo un conjunto de comandos específicos, o incluso un único comando.

Para permitir o restringir el uso de comandos a él o los usuarios que tiene permiso para ejecutar el comando *sudo* se utiliza la siguiente estructura [10]:

nombre_usuario nombre_equipo = (usuario: grupo) comando_restringir/comando_permitir

El significado de cada uno de los términos de esta estructura es el siguiente:

nombre_usuario: El nombre de usuario puede ser un usuario, un alias de usuario o un grupo.

nombre_equipo: Es el nombre del equipo o hostname en el que se va a permitir o restringir los comandos. Sus valores pueden ser todos los equipos (ALL), un solo equipo, un alias de equipos, una dirección IP, etc.

(usuario: grupo): Se especifica los usuarios y los grupos en nombre de los cuales se ejecutarán los comandos. Para poder ejecutar comandos con un usuario y grupo específico se usan los comandos ***sudo* -u** y ***sudo* -g**. Si no se indica ningún usuario ni ningún grupo se usará el usuario *root* y el grupo *root*.

comando_restringir/comando_permitir: Especificación/restricción de los comandos que pueden ejecutar los usuarios que están autenticados como administrador.

Ejemplo del uso de la sintaxis: **ivet localhost = (operator) /usr/bin/mount**

Con esta sentencia se le permite al usuario "**ivet**" que ejecute el comando *mount* como el usuario "operator" en el sistema local (localhost, 127.0.0.1, etc.).

En el caso de restringir sería: **ivet localhost = (operator) !/usr/bin/mount**

Cuando se trata de servidores, se debe habilitar *sudo* para que cada usuario únicamente pueda ejecutar, como administrador, los comandos mínimos necesarios para llevar a cabo su tarea, de ahí que una de las ventajas que posee es la capacidad de hacer seguimiento a los intentos fallidos de acceso *root*.

Ejemplo: Si un usuario estándar, en este caso el usuario *carlos,* intenta ejecutar un comando que requiere privilegios elevados usando el comando *sudo*, por ejemplo, reiniciar el servidor Apache, se obtiene lo siguiente:

```
carlos@ivet-System-Product-Name:/home/ivet/Escritorio$ sudo service apache restart
[sudo] password for carlos:
carlos no está en el archivo sudoers. Se informará de este incidente
```

La respuesta indica que el usuario *carlos* no está en el archivo sudoers y que el incidente será reportado. Estos errores suelen registrarse en /var/log/auth.log, /var/log/messages o /var/log/secure dependiendo del sistema. Por ejemplo, si el usuario administrador necesitara filtrar los mensajes registrados relacionados con el comando *sudo* en busca del error obtenido anteriormente:

```
cat /var/log/auth.log
Obtendría los siguientes resultados:
```

```
Mar 27 14:26:41
ivet-System-Product-Name sudo:       carlos: user NOT in sudoers; TTY=pts/1;
PWD=/home/ivet/Escritorio; USER=root; COMMAND=/usr/sbin/service apache restart
```

Se puede observar que esta entrada de *log* se incluye: información de la fecha y hora de ejecución (Mar 27 14:26:41), nombre del terminal (*ivet-System-Product-Name*), el nombre del usuario que intentó ejecutar el comando con *sudo* (*carlos*), el directorio desde el cual se intentó ejecutar el comando (PWD=/home/ivet/Escritorio), el usuario al que se quería invocar (*root*) y el comando con sus argumentos (COMMAND=/usr/sbin/service apache restart)

1.4.8 Selección del sistema de permisos a utilizar

En las descripciones realizadas a los sistemas y herramientas, la autora llegó a la conclusión que de los estudiados existen dos herramientas que servirán de solución a las necesidades que presenta HMAST, en este caso nos referimos las *Listas de Control de Acceso*, que serán empleadas para asignar permisos específicos y de manera individual sobre archivos y directorios logrando una alta granularidad y seguridad al restringir mayormente los permisos, con el objetivo de lograr un mínimo privilegio en los accesos.

En el caso de los permisos a usuarios para que ejecuten determinadas operaciones sobre el sistema se analizaron los comandos *sudo* y *su*, donde la principal diferencia radica en que el comando *su* se emplea para cambiar de un usuario a otro, generalmente de un usuario estándar al usuario administrador por un tiempo determinado. Para usarlo se requiere conocer la contraseña del administrador, las posibilidades para registrar eventos son más limitadas y una vez que un usuario estándar accede como usuario *root*, no hay control de lo que puede hacer en el sistema. Por su parte *sudo*, permite implementar un control de acceso altamente granulado de qué usuarios ejecutan qué comandos, realizando una dosificada entrega de privilegios temporales para una sola orden, por lo que es la que más se ajusta a las necesidades que presenta la herramienta en estos momentos, por lo que se usará para administrar los permisos de los usuarios sobre el sistema

Para darle solución al objetivo general es necesario tener en cuenta algunos aspectos. Por ello a

continuación se realizará una descripción de la herramienta a la que es necesario incluirle el módulo para el control de acceso a recursos en los sistemas GNU/Linux.

1.5 Descripción de la herramienta HMAST

La herramienta HMAST es una aplicación web que permite la administración de máquinas servidoras. La administración se puede realizar tanto de forma local como remota, esta última a través de conexiones seguras mediante el uso del protocolo SSH. Tiene las funcionalidades necesarias para la gestión de usuarios, tareas programadas y servicios telemáticos [11].

1.5.1 Arquitectura.

La arquitectura de HMAST está basada en N-Capas orientada al dominio, dividida en cinco componentes, como muestra la figura 2. La interacción entre los mismos se realiza a través de interfaces y utilizando inyección de dependencias [11].

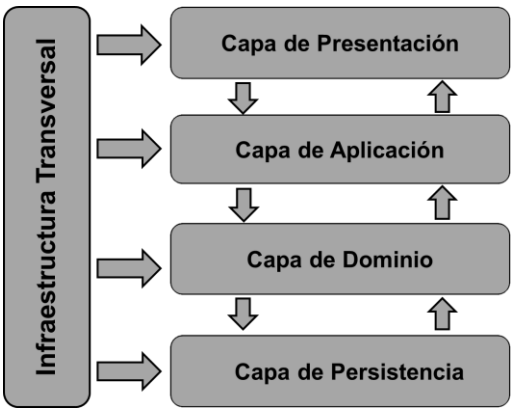


Figura 2. Arquitectura de HMAST

La capa de **presentación** es la que presenta al usuario los conceptos de negocio mediante una interfaz de usuario. La capa de **aplicación** realiza las llamadas a servicios de la capa inferior y tiene la responsabilidad de adaptar la información que le llega, a los requerimientos de los servicios de dominio. La capa de **dominio** es responsable de las validaciones, define las interfaces de persistencia a datos

(contratos de repositorio) pero no los implementa y está compuesta por entidades del dominio que representan objetos del dominio y están definidas fundamentalmente por su identidad, servicios de dominio que contienen la lógica que trata a las entidades como un todo y los contratos de repositorios que son interfaces que especifican las operaciones que deben implementar los repositorios. La capa de **persistencia** es responsable de contener el código necesario para persistir los datos, contiene como componente los repositorios que son clases que implementan los contratos de repositorios definidos en la capa de dominio. Finalmente, la capa **Infraestructura transversal** es responsable de promover la reutilización de código, ella albergará las operaciones de seguridad, autenticación, monitoreo del sistema, mecanismos de persistencia reutilizables, validadores genéricos y todas aquellas operaciones que se puedan utilizar desde otras capas [11].

1.5.2 Funcionalidades que ofrece la herramienta HMAST

La herramienta cuenta con las siguientes funcionalidades [11]:

1. Gestión de servidores lógicos: permite la adición, edición y eliminación de los datos de un servidor lógico, además la conexión remota y desconexión a un servidor seleccionado.
2. Gestión de servicios telemáticos asociados a un servidor lógico: permite la adición, edición y eliminación de los datos de un módulo, así como activación y desactivación de los mismos.
3. 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).

1.5.3 Consideraciones para implementar un módulo para HMAST

Para la implementación de un módulo que se desee integrar a esta herramienta se debe tener en cuenta los siguientes [11]:

- 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, llamadas a componentes de infraestructura para que realicen tareas complementarias.

- Se debe garantizar que no se envíen hacia y desde la capa de presentación objetos de dominio, en su lugar deben viajar Objetos de Transferencia de Datos (*DTO* o *Data Object Transfer* por sus siglas en inglés).
- Las clases de servicios deben ser las únicas responsables (vías de acceso) de acceder a los repositorios, no se puede 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.

Es importante que todo el código reutilizable por más de un repositorio se ponga a disposición de todos en la capa de infraestructura transversal.

1.6 Herramientas, tecnologías y metodología para el desarrollo.

Para lograr uniformidad entre el módulo de gestión del control de acceso a recursos por usuarios del sistema y HMAST se utilizarán las mismas herramientas, tecnologías y la metodología de desarrollo de *software* empleadas en esta última. A continuación, se describirán cada una de estas.

1.6.1 Framework de desarrollo

En el desarrollo de *software*, un framework es una estructura de soporte definida en la cual otro proyecto de *software* puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, librerías y un lenguaje de scripting entre otros *software*s para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Un framework agrega funcionalidad extendida a un lenguaje de programación, esta automatiza muchos de los patrones de programación para orientarlos a un determinado propósito. Un framework proporciona una estructura al código y hace la programación más fácil, convirtiendo complejas funciones en sencillas instrucciones [12].

Spring Framework: 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 de negocios [13]. En el desarrollo del módulo se utilizará la versión 3.2.2.

1.6.2 Lenguaje de programación

Un lenguaje de programación consiste en todos los símbolos, caracteres y reglas de uso que permiten a las personas "comunicarse" con las computadoras. Existen varios cientos de lenguajes y dialectos de programación diferentes. Algunos se crean para una aplicación especial, mientras que otros son herramientas de uso general más flexibles que son apropiadas para muchos tipos de aplicaciones. En todo caso, los lenguajes de programación deben tener instrucciones que pertenecen a las categorías ya familiares de entrada/salida, cálculo/manipulación de textos, lógica/comparación y almacenamiento /recuperación [14].

Java: es un lenguaje de programación orientado a objetos. Es la tecnología subyacente que permite el uso de programas punteros como herramientas, juegos y aplicaciones de negocios. Entre sus principales características se encuentran: lenguaje simple, portable, distribuido, seguro, dinámico, alto rendimiento, entre otras. [15].

Posee una biblioteca denominada **JQuery,** creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX[8] a páginas web. Es *software* libre y de código abierto, posee un doble

[8] *AJAX: A*crónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications).

licenciamiento bajo la Licencia MIT[9] y la Licencia Pública General (GPL) de GNU v2, permitiendo su uso en proyectos libres y privados. Esta al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio. Actualmente es la biblioteca de JavaScript más utilizada [16].

1.6.3 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 proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, PHP, Python, Java[10], C#, Delphi, Visual Basic, etc. 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 [17].

IntelliJ IDEA: es un Ambiente de Desarrollo Integrado (IDE) para el desarrollo de programas informáticos, desarrollado por JetBrains (anteriormente conocido como IntelliJ), y está disponible en dos ediciones: Community Edition y edición comercial. IntelliJ IDEA no está basada en Eclipse como MyEclipse u Oracle Enterprise Pack para Eclipse [18]. En el desarrollo de este módulo se utilizará la versión 14.1.3.

1.6.4 Herramientas CASE

Las Herramientas CASE[11] son un conjunto de herramientas y métodos asociados que proporcionan asistencia automatizada en el proceso de desarrollo del *software* a lo largo de su ciclo de vida. Fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de *software* [19].

[9] *Licencia de software que se origina en el Instituto Tecnológico de Massachusetts.*
[10] *Java: Lenguaje de programación orientado a objetos*
[11] *CASE: Herramienta de Ingeniería de Software Asistida por Computadora, del inglés Computer Aided Software Engineering.*

Visual Paradigm: Visual Paradigm para UML[12] es una herramienta de diseño de *software* diseñado para proyectos de *software* ágil. Soporta varios estándares de modelado de entre los cuales destaca UML. Facilita la creación de *software* y sistemas que se destacan en la experiencia del usuario mediante el apoyo a la utilización eficaz de identificación de casos, recopilación de requisitos, el flujo de los acontecimientos, generación de especificación de requisitos, entre otros [20]. Para el desarrollo del módulo propuesto se utilizará *Visual Paradigm for UML* en su versión 8.0.

1.6.5 Herramienta de diseño gráfico

Las herramientas de diseño gráfico son aquellas que se utilizan en el ámbito profesional del diseño gráfico y que manejan los diseñadores para la elaboración de un trabajo, teniendo en cuenta las especificaciones del cliente. Las mismas utilizan componentes como: el color, la imagen, la tipografía, la fotografía, el equilibrio, el contraste, entre otros [21].

ForeUI: es una herramienta de creación de prototipos, diseñada para crear maquetas, wireframes y prototipos para cualquier aplicación o sitio web. Con ForeUI, el estilo del prototipo se puede cambiar cambiando el tema de la interfaz de usuario. El comportamiento de los prototipos se puede diseñar mediante la definición de diagramas de flujo para manejar. El prototipo puede ser exportado a imágenes wireframe, documentos PDF o simulación HTML5. Esto hace que ForeUI sea una herramienta para compartir ideas, revisar conceptos de diseño, recopilar comentarios y pruebas de usabilidad. Funciona en plataformas Windows, Mac OS X, GNU/Linux y Solaris [22]. Para el desarrollo del módulo se utilizará en su versión 4.30 SP1.

1.7 Metodología de desarrollo de *software*

La metodología 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 comprende los procesos a seguir sistemáticamente para idear, implementar y mantener un producto de

[12] *UML: Lenguaje Unificado de Modelado, del inglés Unified Modeling Language.*

software desde que surge la necesidad del producto hasta que se cumple el objetivo por el cual fue creado, es decir define quién debe hacer qué, cuándo y cómo debe hacerlo para obtener los distintos productos parciales y finales. Se clasifican en dos tipos: tradicionales y ágiles [23].

Para el desarrollo del módulo para el control de acceso a recursos por usuarios de los sistemas GNU/Linux, se emplea la metodología UCI, resultante de una variación de la metodología ágil AUP.

AUP[13]-UCI: Esta metodología 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 [24].

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 para el desarrollo del módulo se emplea el escenario número cuatro, el cual indica que proyectos que no modelen negocio solo pueden modelar el sistema con HU[14] [25].

De forma general se siguen utilizando los productos de trabajos definidos en el Expediente de proyecto, algunos son obligatorios independientemente del tipo de proyecto y otros son opcionales en base a las particularidades del mismo.

Conclusiones Parciales

- En el desarrollo de este capítulo fueron analizados los sistemas que se utilizan en la gestión de

[13] *Proceso Unificado Ágil de Scott Ambler o Agile Unified Process (AUP)*

[14] *HU: Historia de Usuario*

permisos a usuarios para el uso seguro de los recursos del sistema, estudio que permitió identificar las características que poseen, el modo de funcionamiento, así como la importancia y riegos del uso que tienen los mismos.

- Se estudiaron diversas herramientas que permiten la configuración de permisos sobre archivos, directorios y los sistemas GNU/Linux para el diseño del módulo.
- Se caracterizó la herramienta HMAST para identificar los aspectos a tener en cuenta para integrar el módulo a esta herramienta.
- La selección de la arquitectura, la metodología, las herramientas y tecnologías a emplear en el diseño e implementación, demostró las potencialidades de todas estas tecnologías, en aras de lograr la creación del módulo propuesto en la presente investigación.

Capítulo 2. Análisis y Diseño

A partir de las características del sistema base HMAST y los elementos planteados por la metodología AUP-UCI, se describe la propuesta de diseño del módulo a desarrollar. Para ello, se especifican las funcionalidades del sistema mediante los requisitos funcionales y las historias de usuario, y se describe la arquitectura y los patrones de diseño que se emplean.

2.1 Propuesta de solución

En la presente investigación se propone el desarrollo de un módulo que contiene funcionalidades que permitirá gestionar el control de acceso a los recursos en los sistemas operativos GNU/Linux desde HMAST.

El módulo permitirá asignar, modificar o denegar, los correspondientes permisos a los usuarios del sistema GNU/Linux, que así lo requieran para tener acceso a los archivos o directorios y a ejecutar determinadas operaciones en el sistema sin vulnerar la seguridad. Para ello el usuario administrador deberá iniciar sesión en la herramienta, agregar un nuevo servidor lógico y cargar el módulo de control de acceso, así una vez dentro podrá listar todos los usuarios y grupos que se encuentran en el servidor. En ambos casos se seleccionará uno de ellos y se podrá obtener una lista de los archivos y directorios que posee, así como los permisos que tiene para acceder a ellos, permitiendo al administrador modificarlos ya sea para leerlos, escribir sobre ellos o simplemente ejecutarlos.

Para la ejecución de operaciones sobre el sistema, los privilegios serán gestionados (ya sea permitir o revocar) a través de la modificación del archivo /etc/sudoers/ por el administrador, donde serán añadidos los usuarios que así lo requieran con sus especificaciones de cuáles comandos podrán ejecutar como administradores o en nombre de otros usuarios o grupos del sistema. En el caso de que ya existan usuarios con permisos en el archivo *sudoers*, se mostrará una lista con estos, y si se desea modificar los permisos generales que ya tiene o los comandos que podrán ejecutar solo habrá que seleccionar el usuaria deseado de la lista mostrada.

El módulo proporcionará a través de una interfaz gráfica de usuario en HMAST, la funcionalidad de gestionar el control de acceso a los recursos del sistema GNU/Linux, lo que permitirá a los administradores de la herramienta asignarle los permisos necesarios a los usuarios y grupos del sistema.

2.2 Artefactos generados

El proceso de desarrollo es guiado por la metodología AUP-UCI. Teniendo en cuenta que no se modela el negocio y se ajusta, por tanto, al escenario 4 que establece esta metodología, las funcionalidades se describen en un documento de Especificación de Requisitos de *Software*. Estos requisitos son encapsulados mediante Historias de Usuario, lo cual constituye el principal artefacto generado durante el diseño del módulo [26].

2.2.1 Especificación de requisitos

El Glosario de Terminología Estándar de Ingeniería de *Software* define al requisito como una condición que necesita un usuario para resolver un problema o lograr un objetivo. También como una capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente [27]. El propósito de su gestión es establecer un entendimiento común entre el usuario y el desarrollador. Los requerimientos se clasifican en requerimientos funcionales y no funcionales.

- **Requisitos funcionales**

Los requerimientos funcionales son la determinación exacta de qué debe ser capaz de hacer el sistema, estas se corresponden con opciones que ejecutará el *software*, operaciones realizadas de forma oculta o condiciones extremas a determinar por el sistema [27].

Tabla 1. Listado de Requisitos Funcionales

Asignado a	**Ítem ***	**Descripción**
Prioridad		**Alta**

Ivet Campos Cesar Desarrollador	RFN1	Modificar permisos de lectura, escritura y ejecución a un usuario sobre archivos.
Ivet Campos Cesar Desarrollador	RFN2	Modificar permisos de lectura, escritura y ejecución a un usuario sobre directorios.
Ivet Campos Cesar Desarrollador	RFN3	Modificar permisos de lectura, escritura y ejecución a un grupo sobre archivos.
Ivet Campos Cesar Desarrollador	RFN4	Modificar permisos de lectura, escritura y ejecución a un grupo sobre directorios.
Prioridad		**Media**
Ivet Campos Cesar Desarrollador	RFN5	Adicionar usuario estándar al archivo de configuración /etc/sudoers con los permisos para administrar del sistema.
Ivet Campos Cesar Desarrollador	RFN6	Eliminar usuario estándar del archivo de configuración /etc/sudoers.
Ivet Campos Cesar Desarrollador	RFN7	Editar los permisos de un usuario que se encuentre en el archivo de configuración /etc/sudoers.
Ivet Campos Cesar Desarrollador	RFN8	Editar los comandos que puede ejecutar un usuario.
Ivet Campos Cesar Desarrollador	RFN9	Establecer permisos de ejecución a un usuario del sistema.
Ivet Campos Cesar Desarrollador	RFN10	Revocar permisos de ejecución a un usuario del sistema.
Prioridad		**Baja**
Ivet Campos Cesar Desarrollador	RFN11	Listar los usuarios del sistema.
Ivet Campos Cesar Desarrollador	RFN12	Listar los grupos del sistema.
Ivet Campos Cesar Desarrollador	RFN13	Listar los archivos y directorios de un usuario y los permisos que tienen sobre estos.
Ivet Campos Cesar Desarrollador	RFN14	Listar los usuarios que pertenecen al archivo de configuración /etc/sudoers.

- **Requisitos no funcionales**

Los requisitos no funcionales se refieren a cualidades que imponen restricciones en el diseño y la implementación [27]. La metodología AUP-UCI propone una taxonomía partiendo de la ISO 25010 donde se asocian estos requisitos a atributos de calidad. Diversos requisitos no funcionales son heredados de la herramienta HMAST, puesto que condicionan el funcionamiento del módulo para lograr una correcta integración con el sistema base. A continuación, se muestra el listado de los requisitos no funcionales.

Tabla 2. Listado de Requisitos No Funcionales

No.	Nombre del Requisito No Funcional	Atributos de Calidad	Descripción
1	Emplear como lenguaje de programación Java.	Funcionalidad	Estas son restricciones heredadas de HMAST, el módulo debe cumplirlas para poder integrarse correctamente.
2	El módulo se ejecutará sobre el sistema operativo GNU/Linux Nova.		
3	Disponer en el sistema operativo GNU/Linux de los siguientes paquetes: *augeas, augeastools, libjna-java, openjdk-7-jdk.*		
4	Proteger información y los datos, para que personas o sistemas desautorizados no puedan leer o modificar los mismos, y las personas o sistemas autorizados tenga el acceso a ellos.	Seguridad (Acceso restringido)	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.
5	Permitir al usuario un mejor entendimiento y aprendizaje de su aplicación.	Usabilidad	Internacionalizar la información que se muestra, en los idiomas español e inglés.
6	El módulo debe mantener un nivel de ejecución o desempeño especificado en caso de fallos del *software* o de infracción de su interfaz especificada.	Confiabilidad (Tolerancia a fallos)	Ante el fallo de una funcionalidad del sistema el resto de las funcionalidades que no dependen de esta, deberán seguir funcionando.

2.3 Historias de usuario (HU).

Las Historias de Usuario guían la construcción de las pruebas de aceptación y son utilizadas para estimar tiempos de desarrollo. En este sentido, proveen los detalles suficientes para hacer una estimación razonable del tiempo que llevará implementarlas. Además, se deben detallar a través de la comunicación con el cliente, pues constituyen una base para las pruebas funcionales [28]. A continuación, se muestran las siguientes Historias de Usuarios, el resto pueden encontrarse en el Anexo 1 del documento en su versión digital.

1. Modificar permisos de lectura, escritura y ejecución a un usuario sobre archivos.
2. Adicionar usuario estándar al archivo de configuración /etc/sudoers con todos los permisos para administrar el sistema.
3. Listar todos los usuarios del sistema.

HU: Modificar los permisos de lectura, escritura y ejecución a un usuario sobre archivos.

Número: 1	**Nombre de la Historia de Usuario:** Modificar los permisos de lectura, escritura y ejecución a un usuario sobre un archivo.
Programador: Ivet Campos Cesar	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 1 semana
Riesgo en Desarrollo: Alto	**Tiempo Real:** 1 semana
Descripción: Permite modificar los permisos de lectura, escritura y ejecución ya existentes que posee un usuario sobre un archivo seleccionado.	
Acciones para lograr el objetivo (precondiciones y datos): La funcionalidad comienza cuando se selecciona un usuario de la lista y se elige el botón de editar que muestra una interfaz donde se editarán los permisos de los archivos seleccionados. • **Nombre de la ventana**: Editar permisos. • **Buscar:** Se buscarán los archivos a los cuales se les modificar los permisos. • **Cargar:** Se muestra una interfaz donde se despliega la lista de archivos y directorios del sistema y se selecciona el archivo al que se desea modificar los permisos y se añade a la lista de archivos. • **Opción para añadir a la lista de archivos:** Agrega los archivos seleccionados. • **Opción para eliminar de la lista de archivos:** Elimina los archivos seleccionados.	

- **Permisos:** Se selecciona los permisos por los que serán modificados los que posee el archivo.
- Un botón para **aceptar** la modificación realizada.
- Un botón para **cancelar** las acciones de modificación.
- Un botón para **cerrar** la ventana de edición.

Flujo: La funcionalidad tiene lugar en la interfaz de Permisos de usuarios.

Observaciones:

Prototipo de interfaz:

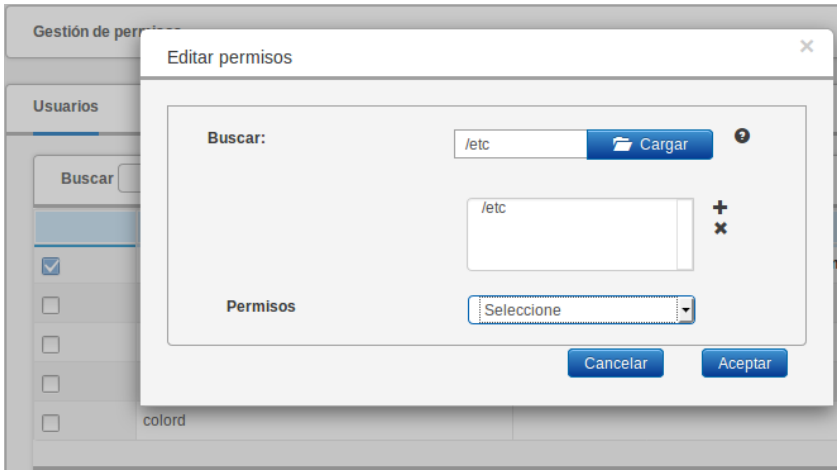


Figure 3. Buscar archivo

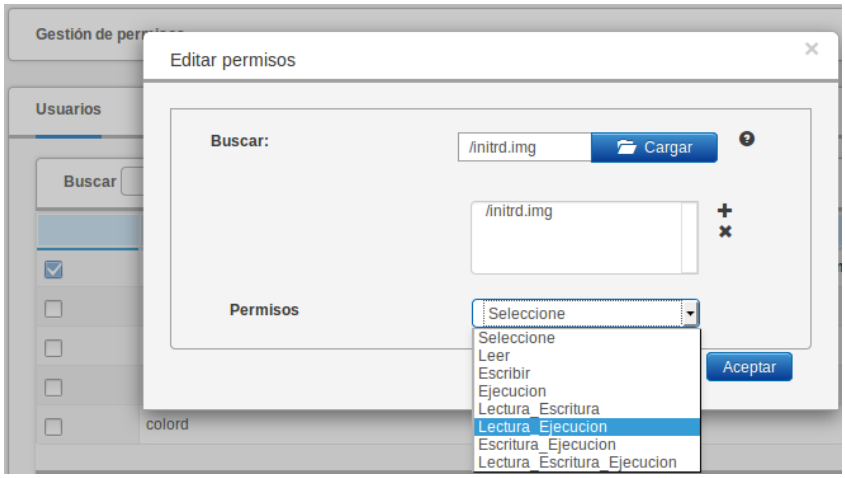


Figure 4. Modificar permisos de archivo

HU: Adicionar usuario estándar al archivo de configuración /etc/sudoers con los permisos para administrar el sistema.

Número: 5	**Nombre de la Historia de Usuario:** Adicionar usuario estándar al archivo de configuración /etc/sudoers con los permisos para administrar el sistema
Programador: Ivet Campos Cesar	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:** 18
Riesgo en Desarrollo: Medio	**Tiempo Real:** 18

Descripción: Permite adicionar un usuario seleccionado al archivo de configuración */etc/sudoers* con los permisos de administración sobre el sistema.

Acciones para lograr el objetivo (precondiciones y datos):
La funcionalidad comienza cuando se selecciona el botón de añadir que muestra una interfaz con los campos obligatorios a llenar para adicionar el usuario en el archivo /etc/sudoers.

- **Nombre de la ventana**: Adicionar especificación de permisos de usuario.
- **Seleccione:** Se selecciona el usuario que tendrá los permisos.
- **Anfitrión (Host):** Se especifican que tendrá permisos en todos (ALL) los anfitriones.
- **Usuarios:** Se especifican que tendrá permisos en nombre de todos (ALL) los usuarios.
- **Grupos:** Se especifican que tendrá permisos en nombre de todos (ALL) los grupos.
- **Comandos:** Se especifican que tendrá permisos para ejecutar todos (ALL) los comandos.
- Un botón para **aceptar** la configuración realizada.
- Un botón para **cancelar** las acciones de configuración.
- Un botón para **cerrar** la ventana de configuración.

Flujo: La funcionalidad tiene lugar en la interfaz de Administradores.

Observaciones:

Prototipo de interfaz:

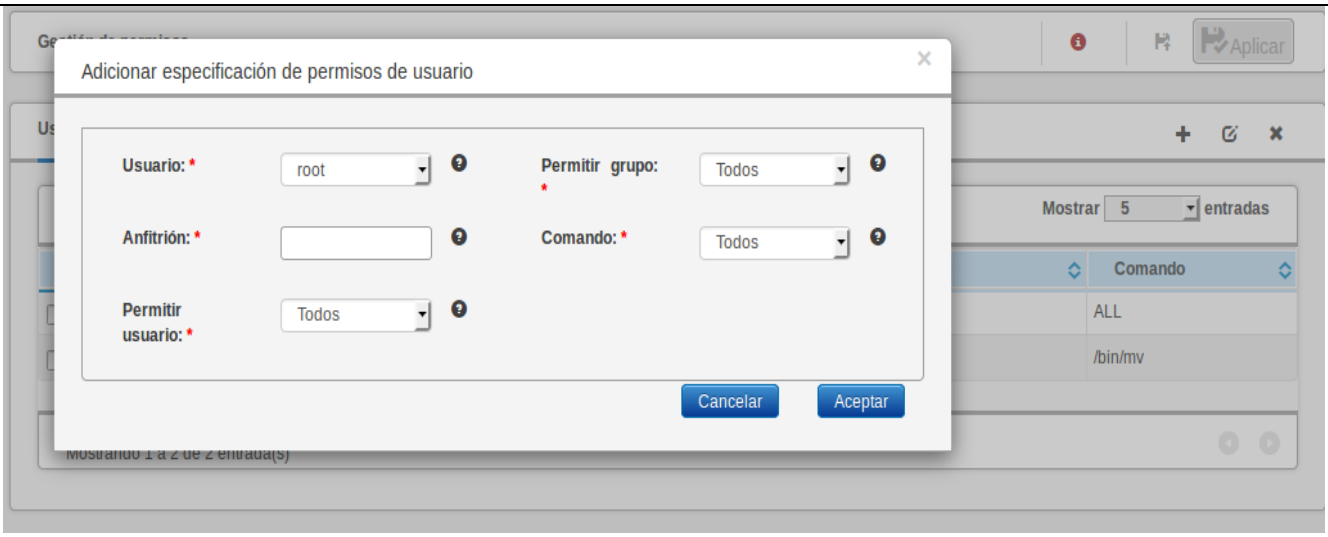


Figure 5. Establecer permisos de administración a un usuario

HU: Listar todos los usuarios del sistema.

Número: 11	**Nombre de la Historia de Usuario:** Listar todos los usuarios del sistema.
Programador: Ivet Campos Cesar	**Iteración Asignada:** 4
Prioridad: Baja	**Tiempo Estimado:** 18
Riesgo en Desarrollo: Bajo	**Tiempo Real:** 19
Descripción: Permite listar los usuarios del sistema junto a la lista de los grupos a los que pertenece.	
Acciones para lograr el objetivo (precondiciones y datos): Se muestran al administrador los siguientes datos de los usuarios: • **Buscar**: Se introduce el nombre del usuario al cual se requiere listar los grupos. • **Listado de grupos:** Muestra los grupos a los que pertenecen el usuario. • Un botón que permite editar aspectos del usuario**.**	
Flujo: La funcionalidad tiene lugar en la interfaz Permisos de usuarios.	
Observaciones:	
Prototipos de interfaz:	

Figure 6. Buscar Usuario

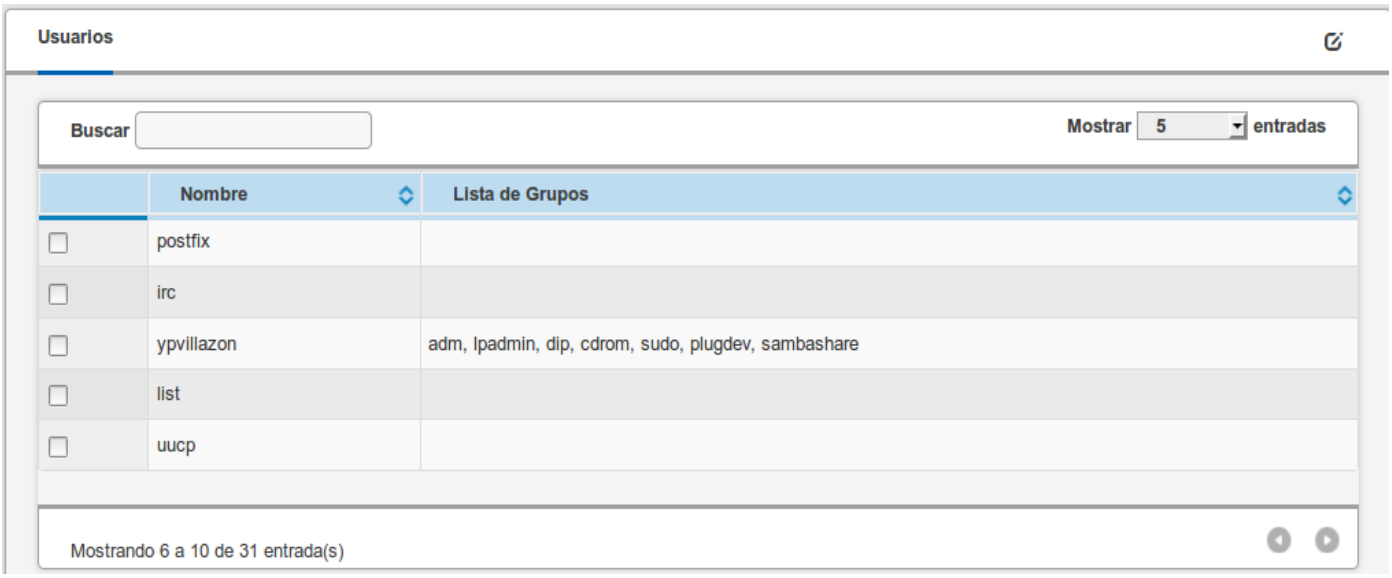


Figure 7. Listado de usuario con sus grupos

2.4 Arquitectura del módulo a desarrollar

En el capítulo anterior se realizó una breve explicación de la arquitectura de HMAST, y debido a las características que esta presenta y por ser este un módulo a integrar a dicha herramienta, se propone que se ajuste a la arquitectura de la misma, incluyendo en cada una de sus capas el módulo *ControlSystem*.

El proceso de diseño de la arquitectura debe decidir cuál funcionalidad es la más importante a desarrollar. Además, define cuáles son los componentes más básicos del sistema y cómo se relacionan entre ellos para implementar la funcionalidad [29].

Se determina como arquitectura del módulo, la establecida para la herramienta base HMAST (ver Figura 11), para lograr la consistencia con los componentes del sistema. Se emplea una arquitectura N-Capas orientada al Dominio, la cual tiene 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 [30].

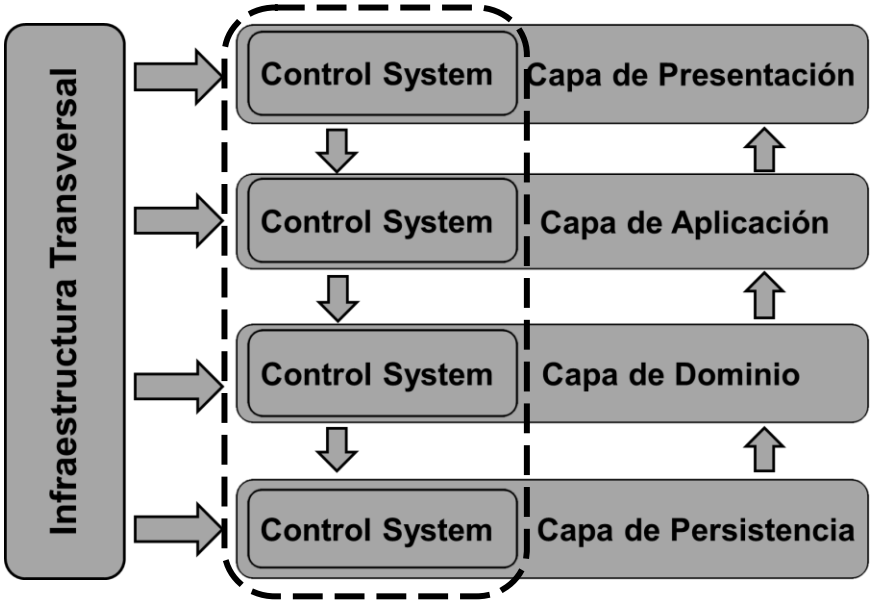


Figure 8. Arquitectura del módulo Control de Acceso

2.5 Diagrama de clases por paquete

El diagrama de paquetes muestra las agrupaciones lógicas en que está dividido el sistema, así como las dependencias entre dichas agrupaciones [30]. Para el diseño del diagrama de paquetes se toma como referencia la arquitectura anteriormente propuesta.

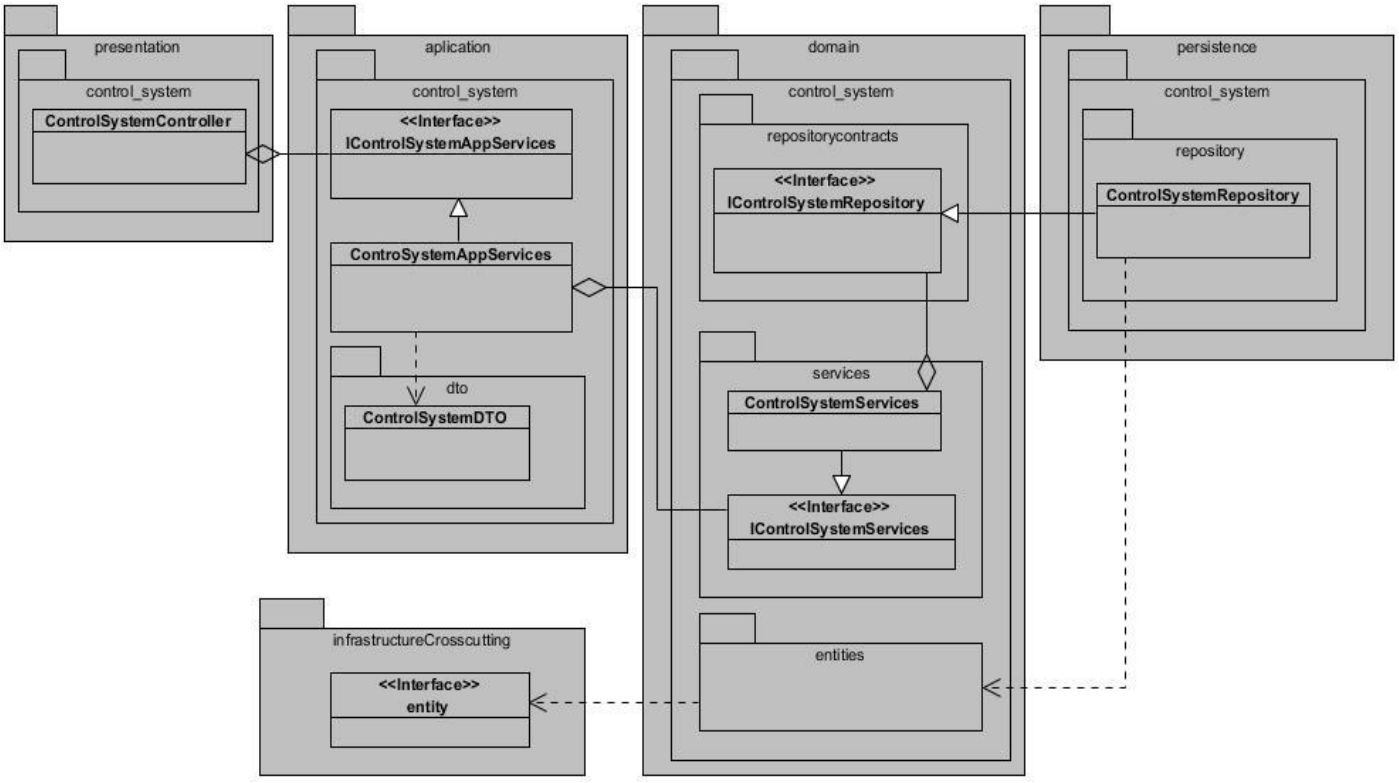


Figure 9. Diagrama de clases por paquete

A continuación, se realiza una descripción del módulo teniendo en cuenta los paquetes que contiene y la organización de sus elementos.

En la figura 10 se representa la capa Aplicación, esta contiene el paquete control_system donde se encuentra la interfaz ***IControlSystemAppService*** que define los métodos que serán invocados en el futuro desde la capa Presentación. También se encuentra la clase ***ControlSystemAppService***, que se encarga de realizar la implementación de los métodos de la interfaz ***IControlSystemAppService***. En ***control_system*** se encuentra también el paquete ***dto***, que contiene los objetos de transferencia de datos. Los objetos de transferencia de datos son enviados desde y hacia la capa de Presentación, la cual depende de ***IControlSystemAppService***. La clase encargada de realizar las conversiones de entidades a

DTO y viceversa es la clase ***ControlSystemAppService***.

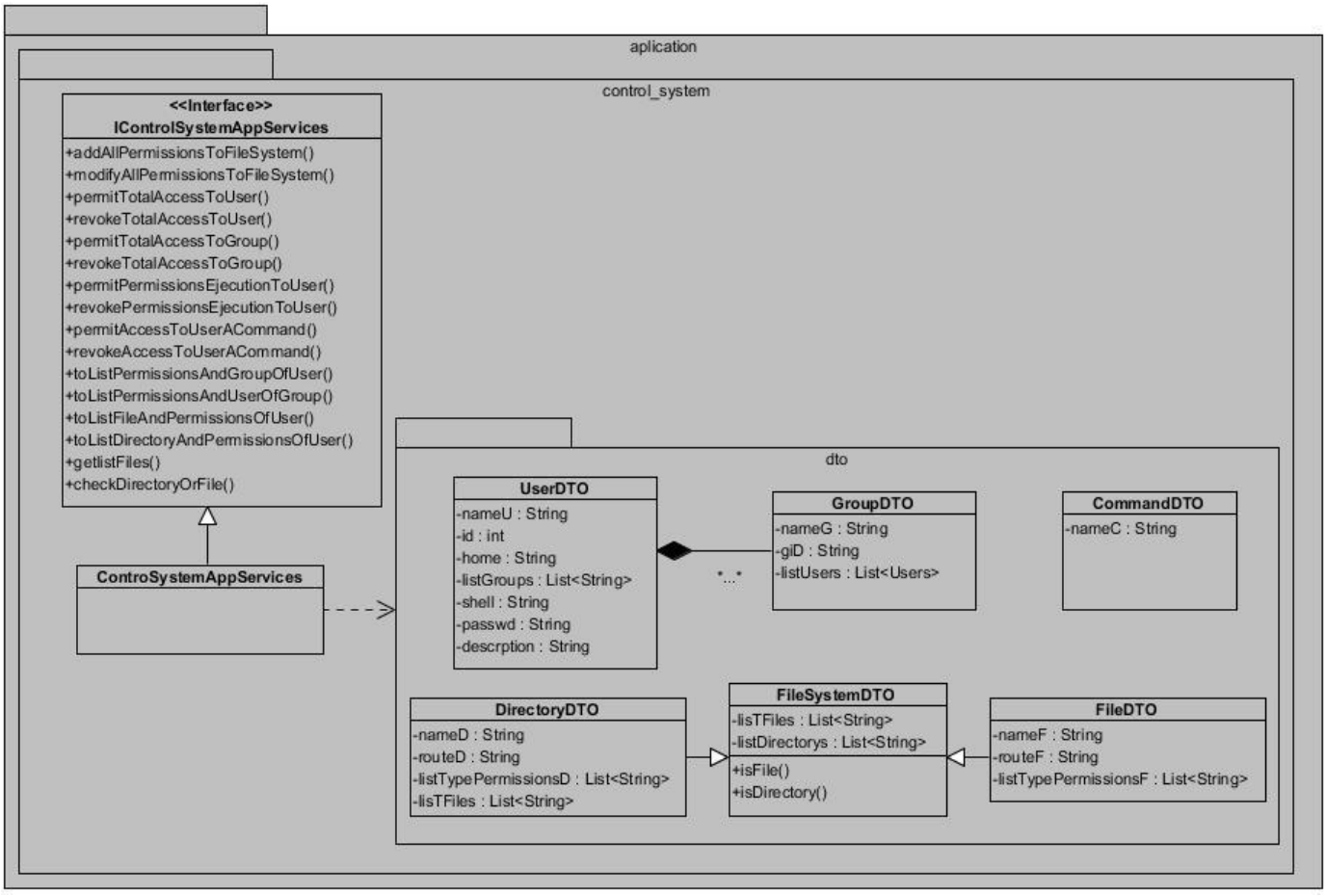


Figure 10. Capa de aplicación

La figura 11 representa la capa de Dominio que se relaciona con la capa de Aplicación a través de una relación de agregación con la interfaz ***IControlSystemService*** mediante la inyección de dependencia. Esta capa contiene dentro de ***control_system***, tres paquetes denominados **entities**, **services** y **repositorycontrats**, los cuales se describen a continuación.

Paquete entities: se definen las entidades del módulo, que representan los objetos del dominio. En la

figura 11 se representan las entidades File, Directory, FyleSystem, User, Group y Command**.** En cada entidad se realizan validaciones atómicas para cada atributo.

Paquete services: se define la interfaz ***IControlSystemService*** y la clase ***ControlSystemService*** que es donde se implementan los métodos definidos en esta interfaz. El objetivo de ***ControlSystemService*** es realizar las validaciones que no se implementen en las entidades. Además, tiene una relación de agregación con ***IControlSystemRepository*** y se realiza mediante una inyección de dependencias.

Paquete repositorycontrats: define los contratos de repositorios en ***IControlSystemRepository***, pero no realiza su implementación.

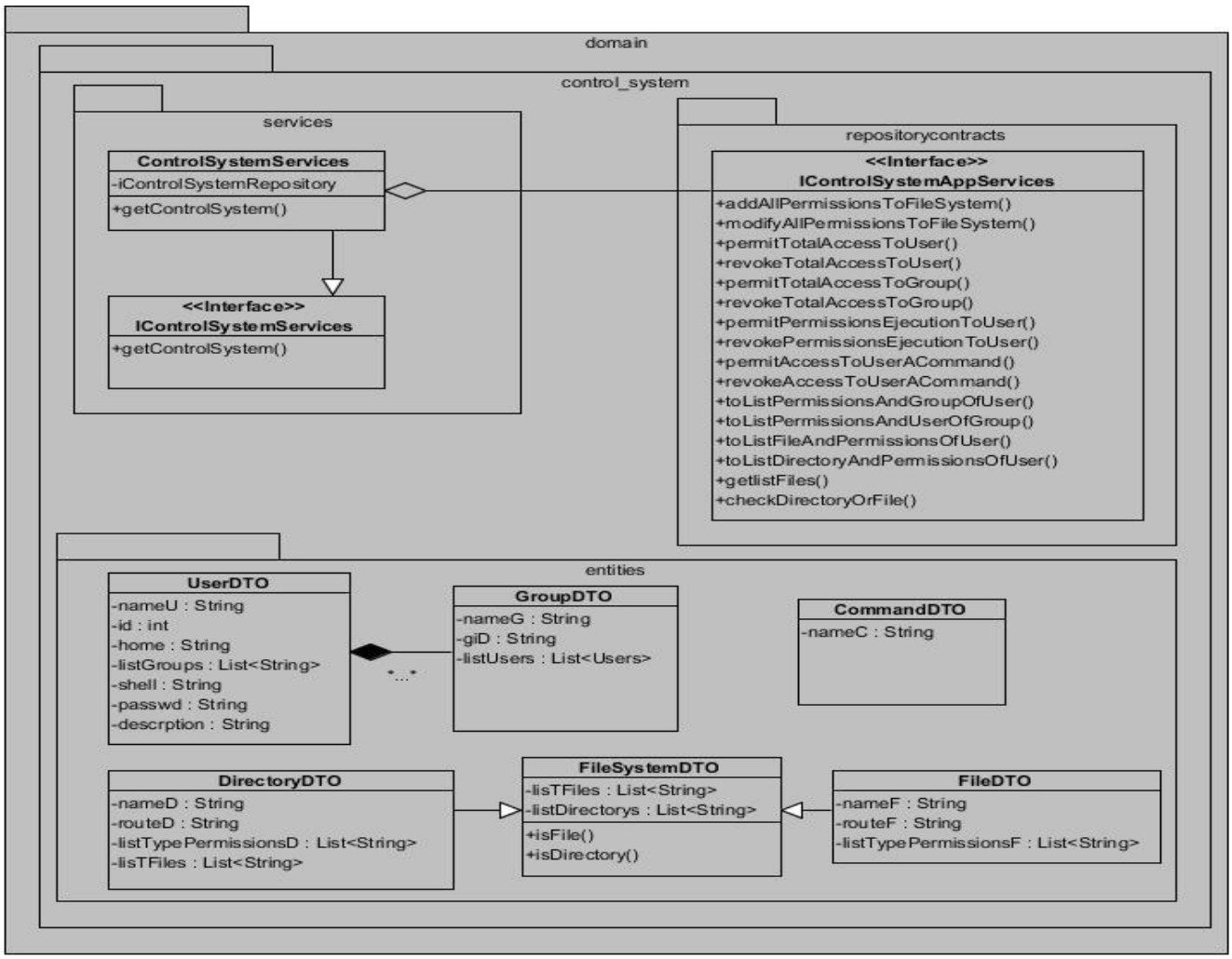


Figure 11. Capa de dominio

En la figura 12 se representa el paquete ***control_system*** correspondiente a la capa Persistencia. Contiene el paquete repository en el cual se encuentra la clase ***IControlSystemRepository***, que se encarga de implementar todos los métodos establecidos en la interfaz ***IControlSystemRepository*** con el objetivo de lograr la persistencia de los datos que se manejan en el módulo.

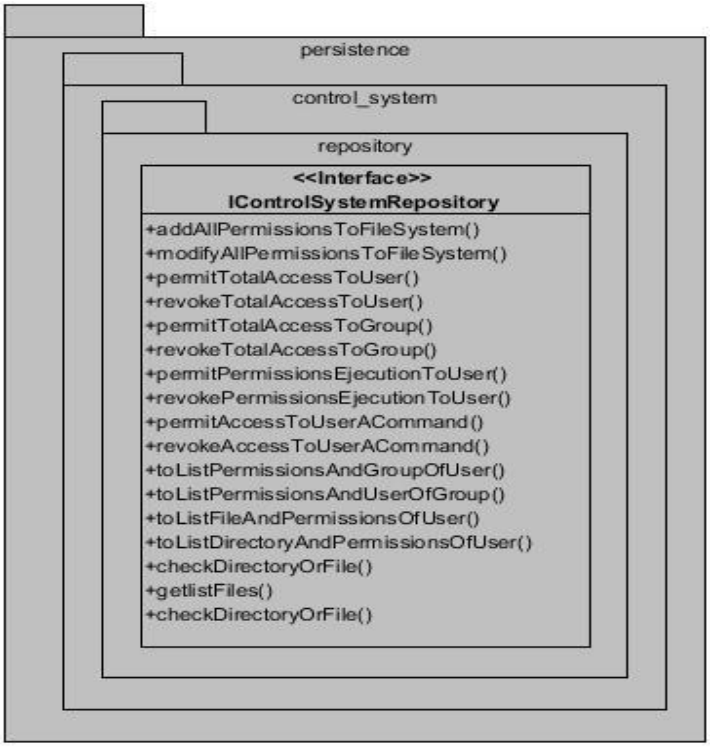


Figure 12. Capa de persistencia

2.6 Patrones de diseño

Los patrones de diseño son principios generales de soluciones que aplican ciertos estilos que ayudan a la creación de *software*. Brindan una solución ya probada y documentada a problemas de desarrollo de *software* que están sujetos a contextos similares [31]. En el modelo de diseño del módulo se aplican patrones de asignación de responsabilidades GRASP[15] y patrones GoF[16], los cuales favorecen un mejor ordenamiento, estructuración y entendimiento del diseño.

[15] *GRASP: Patrones Generales de Software para Asignar Responsabilidades, del inglés General Responsibility Assignment*
[16] *GOF: Patrones de diseño, del inglés Gang of Four, nombre con el que se conoce a los autores del libro Desing Patterns*

2.6.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales del diseño de objetos y la asignación de responsabilidades, expresados como patrones. A continuación, se describen los utilizados en el desarrollo del módulo [31].

- **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.
- **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 pone de manifiesto cuando en la clase *ControlSystemAppService* de la capa de Aplicación se realiza la creación de los DTO (*Objeto de Transferencia de Datos*) 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*.
- **Bajo acoplamiento:** consiste en asignar responsabilidades de manera que el acoplamiento permanezca bajo. 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 *IControlSystemRepository*, y la implementación de sus métodos es realizada por la clase *ControlSystemRepository* en la capa de Persistencia.

- **Alta cohesión:** la cohesión es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento. Las clases tienen una responsabilidad moderada en un área funcional y colaboran con otras clases para llevar a cabo una tarea determinada. Permite que las clases sean fáciles de entender, mantener y reutilizar. Se emplea en la clase *ControlSystemRepository*, que tiene la responsabilidad de interactuar con los objetos y realizar operaciones de asignación y eliminación de permisos.

2.6.2 Patrones GoF

Los patrones GoF son patrones de asignación de responsabilidades y se pueden categorizar en tres grupos teniendo en cuenta su propósito: creacionales, estructurales y de comportamiento [31].

- **Patrón solitario (Singleton):** es un patrón de tipo creacional con el objetivo de garantizar la existencia de una única instancia para una clase y posibilitar el acceso global a dicha instancia. Su empleo se evidencia cuando se realiza una conexión SSH a un servidor, pues con una única instancia del objeto conexión se realizan las operaciones sobre este.

Conclusiones parciales

- A partir del estudio realizado en el capítulo 1 sobre los sistemas que permiten la asignación de privilegios en los sistemas GNU/Linux, se definieron las características y funcionalidades del módulo propuesto, obteniendo veinte requisitos de ellos catorce funcionales y seis no funcionales.
- Se asumió como arquitectura a emplear la N-Capas orientada al Dominio, coincidiendo con la empleada en la herramienta base HMAST lo cual permitió mantener la homogeneidad del módulo con la herramienta y se tomó como base para el diseño del diagrama de clases por paquete.
- Se tuvieron en cuenta varios patrones de diseño que favorecen la reutilización y mantenibilidad de código.

Capítulo 3. Implementación y Pruebas

En este capítulo se describe el proceso realizado para la implementación del sistema, en el que se presentan los estándares de codificación empleados para el desarrollo del módulo, para así mantener la uniformidad con la codificación de los restantes módulos. De un total de catorce requisitos funcionales, solo fueron implementados doce, alcanzando un 80% de implementación, para estos doce se establece la estrategia de pruebas y se documenta la realización de las pruebas seleccionadas. Para comprender la distribución física de los componentes que conforman el sistema, también se modela el diagrama de despliegue.

3.1 Estándar de codificación

Las técnicas de codificación contribuyen a una mejor compresión del código fuente [32]. Los estándares que se emplean en el módulo se ajustan 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[17].

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.

Identación. La unidad de identación 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.

[17] *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 (control_system).

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

kkk → clases o subpaquetes.

3.2 Pruebas de *software*

La prueba de *software* es una actividad realizada para evaluar la calidad de un producto y mejorarlo a través de la identificación de problemas. Consiste en la verificación dinámica del comportamiento de un programa con un conjunto finito de casos de prueba. Las pruebas no aseguran la ausencia de errores, sino que están orientadas a demostrar que existen defectos en el *software* [33].

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 estrategia de pruebas de *software* que se emplea propone pruebas internas y de aceptación, de alto y bajo nivel que incluyen todos los componentes del módulo. Para los doce requisitos que fueron implementados se realizarán pruebas de unidad, pruebas e integración, pruebas funcionales y pruebas de aceptación.

3.2.1 Niveles de prueba

- **Prueba de unidad**

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 [34].

- **Prueba de integración**

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger el módulo probado mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el desafío. [34]

- **Pruebas funcionales**

Las pruebas funcionales tienen como objetivo validar cuando el comportamiento observado del software probado, cumple o no con sus especificaciones. Estas se realizan mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el *software*, con el objetivo de maximizar la producción de las pruebas, es decir, maximizar el número de los errores encontrados por un número finito de los casos de prueba [35].

- **Pruebas de validación**

Las pruebas de aceptación se realizan con el objetivo de validar que el sistema cumple con el funcionamiento esperado y permitir que el usuario determine su aceptación desde el punto de vista de su funcionalidad y rendimiento.

3.2.2 Prueba de unidad

La prueba de unidad que se realiza hace uso del método de caja blanca y de la técnica del camino básico. El método del camino básico permite al diseñador de casos de prueba derivar una medida de complejidad lógica de un diseño procedural y usar esa medida como guía para la definición de un conjunto básico de

caminos de ejecución. Los casos de prueba derivados del conjunto básico garantizan que durante la prueba se ejecute por lo menos una vez cada sentencia del programa [34].

Las unidades de prueba más pequeñas son las operaciones dentro de la clase. A continuación, se realiza la técnica del camino mínimo a la operación *loadFilesSystem* de la clase ***ControlSystemRepository*** que permite obtener la lista de archivos y directorios de un usuario del servidor.

(1)	**public** LinkedList<File> loadFilesSystem(LogicalServer logicalUser, String path) **throws** JSchException, IOException, InterruptedException, EInvalidPath, EInvalidEntity {
(2)	LinkedList<File> listFiles = new LinkedList<File>();
(3)	IdentityGenerator id = new IdentityGenerator();
(4)	**if** (path == null || path.equals("")){
(5)	path = "/";
(6)	}
(7)	IPersist conection = logicalUser.getPersistens();
(8)	PersistOutput persistOutput = conection.executeComandInServerWithCodeOutput("ls -pA " + "'" + path + "'");
(9)	**if** (persistOutput.getCode() == 2) {
(10)	**throw** new EInvalidPath("message.exception.not.exist.path.in.server");
(11)	}
(12)	String files[] = persistOutput.getStandarOutput().split("\n");
(13)	path = (path.equals("/")) ? "" : path;
(14)	**for** (String file : files) {
(15)	**if** (!file.equals("")) {
(16)	**if** (file.contains("/")) {
(17)	listFiles.add(new File(file.substring(0, file.length() - 1), path + "/" + file.substring(0, file.length() - 1), true, false, id.getNewUUID()));
(18)	}

(19)	**else** {
(20)	listFiles.add(new File(file, path + "/" + file, false, false, id.getNewUUID()));
(21)	}
(22)	}
(23)	}
(24)	**return** listFiles; }

Luego de numerar las líneas de código, se diseña la gráfica del programa que describe el flujo de control lógico empleando nodos y aristas.

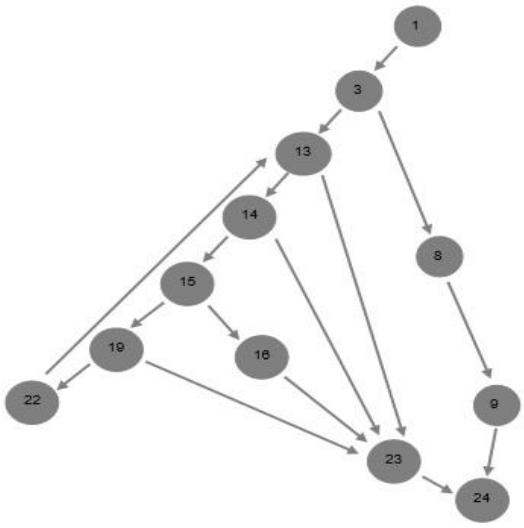


Figure 13. Grafo de Flujo

A partir del grafo obtenido con 15 nodos y 20 aristas se calcula la complejidad ciclomática *V(G)*, la cual constituye una métrica de *software* que proporciona una medida cuantitativa de la complejidad lógica del programa [36].

V(G) = cantidad_aristas – cantidad_nodos + 2

V(G) = 16 – 12 + 2 = 6

Una ruta independiente es cualquier ruta del programa que ingrese al menos un nuevo conjunto de instrucciones de procesamiento o una nueva condición [34]. La cantidad de rutas independientes son establecidas por la complejidad ciclomática, por tanto, se identifican seis rutas, tal y como se muestra en la siguiente tabla.

Tabla 3. Listado de caminos básicos.

No. Ruta	Camino
1	1,3,8,9,24
2	1,3,13,23,24
3	1,3,13,14, 23,24
4	1,3,13,14,15,16,23,24
5	1,3,13,14,15,19,23,24
6	1,3,13,14,15,19,22,13,23,24

El valor de V(G) ofrece además un límite superior del número de pruebas de debe diseñarse y ejecutarse para garantizar la cobertura de todas las instrucciones [34]. Por tanto, se diseñan casos de pruebas para ser aplicados a cada ruta independiente.

Caso de Prueba de Unidad	
No. ruta: 1	**Ruta:** 1,3,8,9,24
Nombre de la persona que realiza la prueba: Ivet Campos Cesar	
Descripción de la prueba: Obtener una lista de archivos y directorios	
Entrada: Se envía como parámetro la ruta o *path* que no existe y que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado.	
Resultado esperado: El sistema debe mostrar una excepción con el mensaje "*No existe la ruta*"	
Evaluación de la Prueba: Satisfactoria. Se obtuvo el mensaje esperado.	

Caso de Prueba de Unidad	
No. ruta: 2	**Ruta:** 1,3,13,23,24

Nombre de la persona que realiza la prueba: Ivet Campos Cesar
Descripción de la prueba: Obtener una lista de archivos y directorios
Entrada: Se envía como parámetro la ruta o *path* que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado y que además no posee elementos, o sea la lista vacía.
Resultado esperado: El sistema retornar una lista vacía.
Evaluación de la Prueba: Satisfactoria. Se obtuvo lista de archivos del usuario.

Caso de Prueba de Unidad	
No. ruta: 3	**Ruta:** 1,3,13,14,23,24
Nombre de la persona que realiza la prueba: Ivet Campos Cesar	
Descripción de la prueba: Obtener una lista de archivos y directorios	
Entrada: Se envía como parámetro la ruta o *path* que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado y que además no posee elementos, o sea la lista vacía.	
Resultado esperado: El sistema debe retornar una lista vacía.	
Evaluación de la Prueba: Satisfactoria. Se obtuvo lista de archivos del usuario especificado.	

Caso de Prueba de Unidad	
No. ruta: 4	**Ruta:** 1,3,13,14,15,16,23,24
Nombre de la persona que realiza la prueba: Ivet Campos Cesar	
Descripción de la prueba: Obtener una lista de archivos y directorios	
Entrada: Se envía como parámetro la ruta o *path* que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado y tiene una longitud mayor que uno.	
Resultado esperado: El sistema debe recorrer la lista de archivos y directorios que pertenecen al usuario especificado, y retornar una de lista de directorios.	
Evaluación de la Prueba: Satisfactoria. Se obtuvo lista de directorios del usuario especificado.	

Caso de Prueba de Unidad	
No. ruta: 5	**Ruta:** 1,3,13,14,15,19,23,24
Nombre de la persona que realiza la prueba: Ivet Campos Cesar	
Descripción de la prueba: Obtener una lista de archivos y directorios	
Entrada: Se envía como parámetro la ruta o *path* que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado.	
Resultado esperado: El sistema debe recorrer la lista de archivos y directorios que pertenecen al usuario especificado, y retornar una de lista de archivos.	
Evaluación de la Prueba: Satisfactoria. Se obtuvo lista de archivos del usuario especificado.	

Caso de Prueba de Unidad	
No. ruta: 6	**Ruta:** 11,3,13,14,15,19,22,13,23,24
Nombre de la persona que realiza la prueba: Ivet Campos Cesar	
Descripción de la prueba: Obtener una lista de archivos y directorios	
Entrada: Se envía como parámetro la ruta o *path* que conduce hasta el directorio que está desplegando la lista de archivos y directorios que pertenecen al usuario especificado.	
Resultado esperado: El sistema debe recorrer la lista de archivos y directorios que pertenecen al usuario especificado.	
Evaluación de la Prueba: Satisfactoria. Se obtuvo lista de archivos y directorios del usuario especificado.	

Para la ejecución de la prueba de unidad se tuvo en cuenta cómo se llevó a cabo el proceso de desarrollo del sistema, el que se realizó de forma vertical, ejecutándose un total de cuatro iteraciones. En la primera iteración se detectaron cuatro no conformidades; en la segunda, dos no conformidades y en la tercera iteración una no conformidad, las mismas fueron resueltas en cada iteración antes de pasar a la siguiente. Las no conformidades detectadas se encontraban asociadas a errores de validación, tratamiento de excepciones desde el código y la redacción de mensajes de confirmación o error del sistema.

3.2.3 Prueba de integración

La prueba de integración es una técnica para construir la arquitectura del *software*. El objetivo es tomar componentes a los que se aplicó una prueba de unidad y construir una estructura de programa que determine el diseño [34]. Asumiendo la arquitectura por capas empleada y que el módulo debe incorporarse a una herramienta base, se emplea 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 [34]

3.2.4 Pruebas de funcionalidad

Para esta se escoge el método de caja negra y la técnica de partición equivalente debido a que los niveles de pruebas escogidos pretenden probar el funcionamiento de la interfaz y de las funcionalidades del sistema. Se usará el método de caja negra ya que permite comprobar el comportamiento del *software* a través de los requisitos funcionales [37], obviando el funcionamiento interno y la estructura del programa.

Las pruebas de caja negra pretenden encontrar estos tipos de errores:

- Funciones incorrectas o ausentes.
- Errores en la interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de comportamiento o desempeño.
- Errores de inicialización y de terminación.

Figure 14. Método de Caja Negra

Técnica de prueba: Partición equivalente

Características

- Divide el dominio de entrada de un programa en clases de datos, a partir de las cuales pueden derivarse casos de prueba.
- Descubre clases de errores que, de otra manera, requeriría la ejecución de muchos casos antes de que se observe el error general.
- Reducir al máximo el total de casos de prueba que deben desarrollarse.

El diseño consiste en:

- Identificar clases de equivalencia.
- Crear los casos de prueba.

Clase de equivalencia

Conjunto de estados válidos o inválidos para condiciones de entrada. Las clases de equivalencia se definen de acuerdo a las siguientes directrices:

1. Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válidas.

3. Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y otra no válida.
4. Si una condición de entrada es booleana, se define una clase de equivalencia válida y otra no válida.

3.2.4.1 Diseño de casos de prueba basados en Historias de Usuario.

A continuación, se presentan los casos de prueba que corresponden a la historia de usuario: *Adicionar usuario estándar al archivo de configuración /etc/sudoers con los permisos para administrar del sistema.*

Tabla 4. Casos de prueba

No.	Nombre de campo	Clasificación	Valor nulo	Descripción
1	**Usuario**	Campo de selección	Si	Se habilita solo si el administrador no va a utilizar el nombre de usuario por defecto y desea definir otro usuario de la lista que se va añadir al archivo /etc/sudoers.
2	**Anfitrión**	Campo de texto	Si	Debe contener una cadena de caracteres con un mínimo de 10 elementos, puede contener minúsculas, números y caracteres especiales.
3	**Permitir usuario**	Campo de selección	Si	Se selecciona el usuario de la lista por el cual el usuario definido en el campo **Usuario**, va a tener permisos.
4	**Permitir grupo**	Campo de selección	Si	Se selecciona el grupo de la lista por el cual el usuario definido en el campo **Usuario**, va a tener permisos.
5	**Comando**	Campo de selección	Si	Se selecciona el comando de la lista que el usuario definido en el campo **Usuario**, va a tener permitido ejecutar.

Escenario 1.1 Adicionar usuario con los valores obligatorios.

Descripción: El sistema debe permitir adicionar el usuario con los valores definidos en los campos no nulos.

Flujo central: Al acceder al icono adicionar, se muestra una ventana donde se introducen los valores

válidos, luego se accede al botón aceptar.

Tabla 5. Escenario 1.1 Adicionar usuario con los valores obligatorios

V1	V2	V3	V4	V5
V	V	V	V	V
pepe	Todos	Todos	Todos	Todos
V	V	V	V	V
maría	10.53.3.56	root, pepe1, carlos	Economía, 1205, operador	/bin/sh, /bin/mount, /bin/apt

Respuesta del sistema: El sistema adiciona el usuario y muestra el mensaje: El usuario ha sido adicionado correctamente.

Escenario 1.2 Adicionar usuario con los valores obligatorios vacíos.

Descripción: El sistema no debe adicionar el usuario con los campos obligatorios vacíos.

Flujo central: Al acceder al icono adicionar, se muestra la ventana adicionar usuario, donde se introducen los valores especificados se accede al botón aceptar.

Tabla 6. Escenario 1.2 Adicionar usuario con los valores obligatorios vacíos.

V1	V2	V3	V4	V5
V	V	V	V	V
(vacío)	(vacío)	root	Todos	/bin/sh
V	V	V	V	V
cesol	1qazxsw2.	pepe1, juan	(vacío)	(vacío)

Respuesta del sistema: El sistema muestra un mensaje de error con un aviso de que el usuario no se ha podido adicionar y muestra el parámetro que contiene vacío.

Escenario 1.3 Adicionar usuario con valores obligatorios incorrectos.

Descripción: El sistema no debe permitir adicionar un usuario con valores incorrectos.

Flujo central: Al acceder a la ventana de adicionar, se introducen los valores especificados y se accede al botón aceptar.

Tabla 7. Escenario 1.3 Adicionar usuario con los valores obligatorios incorrectos.

V1	V2	V3	V4	V5
V	V	V	V	V
pepe	Zaq12wsxp232	(vacío)	(vacío)	(vacío)
V	V	V	V	V
carlos	Zaq12wsxpÑ.,23	(vacío)	(vacío)	(vacío)

Respuesta del sistema: El sistema muestra un mensaje de error notificando que el usuario a adicionar contiene valores incorrectos y especifica el o los parámetros que contienen error.

De las pruebas funcionales se realizaron tres iteraciones, detectando nueve, cuatro y tres no conformidades en la primera, la segunda y la tercera iteración respectivamente. De estas no conformidades nueve fueron de errores ortográficos y siete de validación incorrecta.

3.2.5 Prueba de validación.

Las pruebas de aceptación son ejecutadas para validar la conformidad del cliente de acuerdo con las especificaciones definidas. En el caso del módulo para la gestión del control de acceso a recursos por usuarios del sistema GNU/Linux en HMAST, la variable de mayor relevancia es la integridad de los datos, por lo que se define un diseño que garantiza la validación de esta variable. Según la norma ISO/IEC 9126 de la Oficina Nacional de Normalización, el modelo de calidad está definido por categorías, que validan los atributos a partir de seis características: funcionalidad, confiabilidad, usabilidad, mantenibilidad, eficiencia y portabilidad. A partir de la ejecución de las pruebas, el cliente examinó cada una de las funcionalidades con que cuenta el sistema comprobando el correcto funcionamiento de estas y donde pudo determinar, además, que el módulo sí facilita el trabajo de los administradores. Para que así conste, el cliente firma un Acta de aceptación de productos de trabajo, la que figura en el anexo dos del presente trabajo.

3.3 Diagrama de despliegue

El diagrama de despliegue es un tipo de diagrama del Lenguaje Unificado de Modelado que se utiliza para modelar la disposición física de los artefactos *software* en nodos (usualmente plataforma de hardware) [38]. 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.

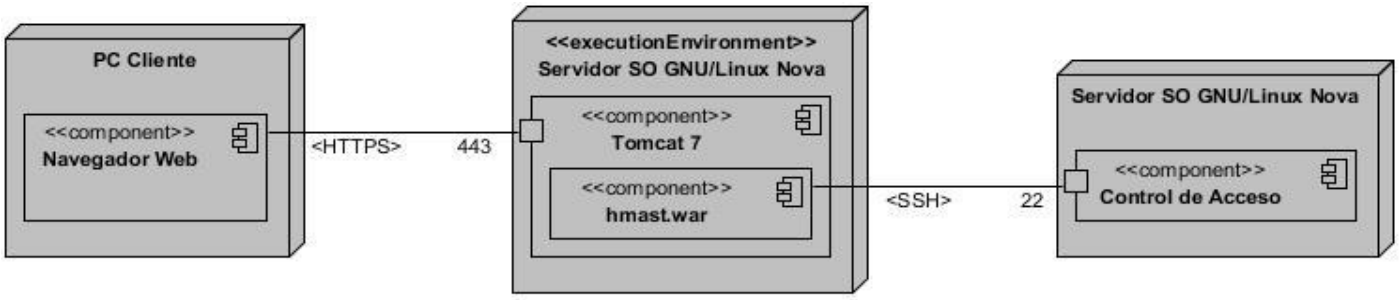


Figure 15. Diagrama de despliegue del módulo

Conclusiones parciales

- El empleo de los estándares de codificación definidos para la herramienta HMAST permitió desarrollar un código reutilizable, de fácil comprensión para su posterior uso.
- De catorce requisitos funcionales especificados, solo se cumplieron doce, por lo que la estrategia de prueba seleccionada fue ajustada a las doce funcionalidades que fueron implementadas permitiendo comprobar las relaciones entre todas las partes del módulo.
- Mediante las pruebas realizadas se verificó que todas las instrucciones del módulo se ejecutan al menos una vez, que los componentes se integran correctamente.
- Se validó que el módulo se ajusta de forma satisfactoria a los requisitos del sistema.

Conclusiones Generales

A partir de la investigación realizada y de los resultados obtenidos referentes al desarrollo del módulo de HMAST para gestionar el control de acceso a recursos del sistema GNU/Linux, se concluye que:

- El estudio del arte permitió identificar las herramientas *Listas de Control de Acceso* y el comando *Sudo* como las más adecuadas para gestionar el control de acceso a recursos en el sistema GNU/Linux.
- Se diseñó el módulo para el control de acceso a recursos por usuarios del sistema lo que permitió generar los requisitos funcionales, los no funcionales y los prototipos de interfaz correspondientes a cada historia de usuario.
- De los catorce requisitos funcionales generados solo fueron implementados doce, lo que representa un 80% de desarrollo del módulo.
- Se realizaron pruebas de *software* a los doce requisitos implementados, lo que permitió comprobar el correcto funcionamiento de estos.

Recomendaciones

Como resultado de la investigación se obtuvo un módulo capaz de realizar la asignación y revocación de permisos a los usuarios de los servidores lógicos sobre los recursos del sistema, por lo que se le debe dar un seguimiento a las siguientes recomendaciones:

- Desarrollar una funcionalidad que permita desde la herramienta HMAST establecer los permisos de ejecución a un usuario del sistema.

Referencias Bibliográficas

[1]. Viera Hernández, Amaury y De La Rosa Gómez, Leonardo. MigrateAD: Migración del Directorio Activo a plataforma libre. 2009.

[2]. Roberto Woo Borrego, *"Seminario: Conceptos generales sobre el Control de Acceso".*

[3]. Administración de usuarios en Linux. [en línea]. 15 noviembre 2016. [Consultado el: 15 noviembre 2016]. Disponible en: https://www.linuxtotal.com.mx/index.php?cont=info_admon_008

[4]. Permisos de archivos y carpetas. [en línea]. 19 noviembre 2016. [Consultado el: 19 noviembre 2016]. Disponible en: http://www.ite.educacion.es/formacion/materiales/85/cd/linux/m1/permisos_de_archivos_y_carpetas.html

[5]. Permisos del sistema de archivos en GNU/Linux. [en línea]. 20 noviembre 2016. [Consultado el: 20 noviembre 2016]. Disponible en: http://www.alcancelibre.org/staticpages/index.php/permisos-sistema-de-archivo

[6]. Listas de Control de Acceso. [en línea]. 29 noviembre 2016. [Consultado el: 29 noviembre 2016]. Disponible en:http://www.alcancelibre.org/ Listas de control de acceso y uso de getfacl y setfacl. - Alcance Libre.htm

[7]. Listas de control de acceso y uso de getfacl y setfacl. [en línea]. 29 noviembre 2016. [Consultado el: 29 noviembre 2016]. Disponible en:http://www.alcancelibre.org/ Listas de control de acceso y uso de getfacl y setfacl. - Alcance Libre.html

[8]. MONOGRÁFICO: Listas de control de acceso (ACL) - Utilización de ACLs en el sistema de archivos | Observatorio Tecnológico. [en línea]. 29 noviembre 2016. [Consultado el: 29 noviembre 2016]. Disponible en: http://recursostic.educacion.es/observatorio/web/es/component/content/article/1065-listas-de-control-de-acceso-acl?start=2

[9]. De la teoría a la práctica: sudo, sudoers y visudo. [en línea]. 5 enero 2017. [Consultado el: 5 enero 2017]. Disponible en: https://docs.bluehosting.cl/tutoriales/servidores/de-la-teoria-a-la-practica-sudo-sudoers-y-visudo.html

[10]. Sudo. [en línea]. 5 enero 2017. [Consultado el: 5 enero 2017]. Disponible en: http://www.alcancelibre.org/staticpages/index.php/como-sudo-linux?query=control+de+acceso

[11]. PÉREZ TASÉ, ALEXANDER. *Módulo de administración del servicio Proxy para HMAST*. La Habana, Cuba: Universidad de las Ciencias Informáticas, 2013.

[12]. Andrés Felipe Gutiérrez. *Kumbia PHP Framework: Porque programar debería ser más fácil.* 2013.: s.n.

[13]. Spring Framework Reference Documentation. [en línea]. 18 diciembre 2016. [Consultado el: 18 diciembre 2016]. Disponible en: http://docs.spring.io/spring/docs/4.1.0.BUILD-SNAPSHOT/spring-framework-reference/htmlsingle/.

[14]. Definición de Lenguaje de programación - ¿Qué es un Lenguaje de programación? [en línea]. 13 enero de 2017. [Consultado el: 13 enero 2017]. Disponible en: http://www.sites.upiicsa.ipn.mx/polilibros/portal/polilibros/p_terminados/PolilibroFC/Unidad_III/Unidad%20III_4.htm

[15]. Introducción de Rubén Venegas en Prezi. [en línea]. 16 enero 2017. [Consultado el: 16 enero 2017]. Disponible en: http://prezi.com/pw8liegicqzi/introducción/.

[16]. Definición de JQuery. [en línea]. 19 enero 2017. [Consultado el: 19 enero 2017]. Disponible en: https://www.ecured.cu/JQuery_UI

[17]. Definición de Entorno de Desarrollo Integrado (IDE). [en línea]. 21 enero 2017. [Consultado el: 21 enero 2017]**.** Disponible en: https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/.

[18]. Definición de IntelliJ IDEA [en línea]. 21 enero 2017. [Consultado el: 21 enero 2017]. Disponible en: https://www.jetbrains.com/idea/

[19]. Herramientas Case. [en línea]. 23 enero 2017. [Consultado el: 23 enero 2017]. Disponible en: http://fds-herramientascase.blogspot.com/.

[20]. Software Design Tools for Agile Teams, with UML, BPMN and More [en línea]. 25 enero 2017. [Consultado el: 25 enero 2017]. Disponible en: http://www.visual-paradigm.com/.

[21]. Herramientas del diseñador gráfico | Catálogo Digital de Publicaciones DC. [en línea]. 25 enero 2017. [Consultado el: 25 enero 2017]. Disponible en: http://fido.palermo.edu/servicios_dyc/publicacionesdc/vista/detalle_articulo.php?id_articulo=4678&id_libro=139.

[22]. Definición de ForeUI. [en línea]. 25 enero 2017. [Consultado el: 25 enero 2017]. Disponible en: https://www.foreui.com

[23]. Laboratorio Nacional de Calidad del Software de INTECO. Ingeniería del Software: Metodologías y ciclos de vida. España, marzo, 2009.

[24]. GONZALEZ SANTIAGO, FELIPE y YOSEL LAZARO VERA GONZALEZ. *Módulo de HMAST Para administración y migración hacia Samba4 del servicio Directorio Activo.*

[25]. Universidad de las Ciencias Informáticas. Metodología de desarrollo para la Actividad productiva de la UCI. 2015.

[26]. S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Sexta edición.

[27]. IEEE-STD-830-1998: PRÁCTICA RECOMENDADA PARA LAS ESPECIFICACIONES DE REQUISITOS DEL SOFTWARE. 2008

[28]. Reidiel Castillo Arbelo y Pablo Soria Acosta. *Herramienta para la Migración y Administración de Servidores (HMAS).*

[29]. Zorrilla Castro, César. Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0… 2010. ISBN 978-84-936696-3-8.

[30]. Gamma, Eric; Helm, Richard; Johnson, Ralph; Vlissides, John. Design Patterns - Elements of Reusable Object-Oriented Software.

[31]. Larman, Craig. UML y patrones, 2da Edición.

[32]. MSDN. Técnicas de codificación. [en línea]. 10 mayo 2017. [Consultado el: 10 mayo2017]. Disponible en: http://msdn.microsoft.com/es- es/library/aa291593(v=vs.71).aspx

[33]. IEEE. Swebok - Guide to the Software engineering body of knowledge. 2004. ISBN 0-7695-2330-7.

[34]. S. Pressman, Roger. Ingeniería de software. Un enfoque práctico. Quinta edición.

[35]. Gestión de las Pruebas Funcionales. [en línea]. 18 mayo 2017. [Consultado el: 18 mayo2017]. Disponible en: http://www.ces.com.uy/documentos/imasd/CES-PRIS_Gestion_Testing.pdf

[36]. De la Puente, Juan Antonio. Fiabilidad y tolerancia a fallos. DIT/UPM. 2001.

[37]. Pressman, Roger (2002). Ingeniería del *Software*, un enfoque práctico, Oc-Graw Hill página 182, consultado 20 de mayo de 2017

[38]. OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, p. 202. [en línea]. 10 mayo 2017. [Consultado el: 10 mayo 2017]. Disponible en: http://doc.omg.org/formal/2007-11-02.pdf