Universidad de las Ciencias Informáticas
Facultad 1

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Título: Módulo para monitorizar la integridad de los archivos para HMAST.

Autora:
Yanelsi Marén Barrera.

Tutores:
Ing. Yordanis Cabreja Núñez.
Ing. Yurenia Hernández Blanco.

Consultante:
Ing. Reidiel Castillo Arbelo.

La Habana.
Junio, 2013.

Declaración de Autoría.

Declaro ser la única autora de este trabajo y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

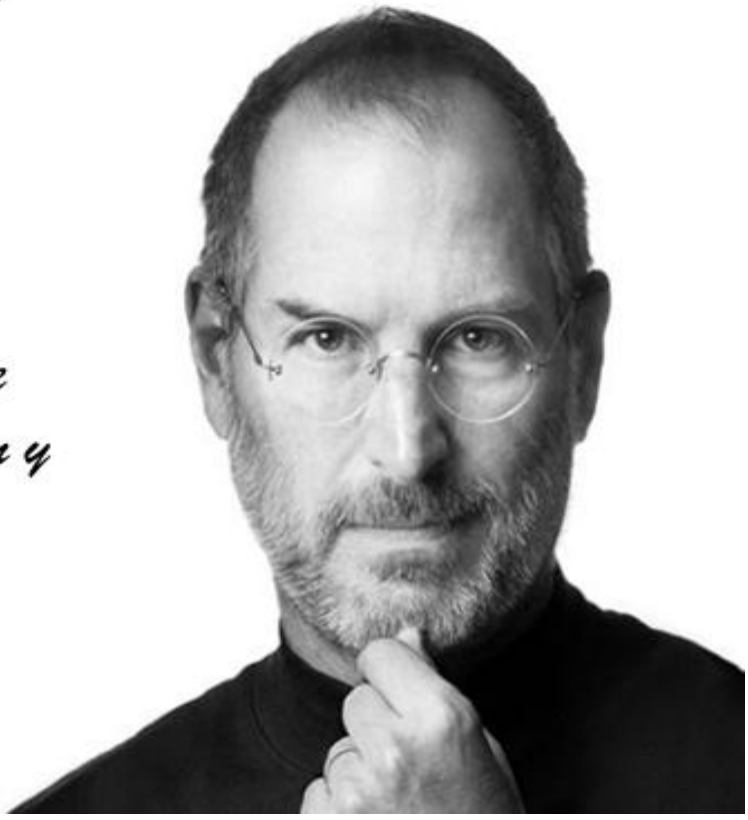
Para que así conste firma la presente a los ___ días del mes de ___ del año 2013.

_____
Yanelsi Marén Barrera

_____ _____
Ing. Yordanis Cabreja Núñez Ing. Yurenia Hernández Blanco

Steve Jobs

Hay que considerar que ponemos nuestro corazón y nuestra alma en estos productos.



Agradecimientos.

Agradecer profundamente a mis padres, por inspirarme a recorrer este camino y acompañarme en su travesía cada minuto.

A mis hermanos Yadira y Vladimir por brindarme su apoyo siempre que lo necesité y su cariño incondicional.

A mi mejor amiga, o mejor dicho mi otra hermana Marielbys por todos los momentos que pasamos juntas.

A mis tutores y mi consultante en especial a Yurenia y a Reidiel por apoyarme y confiar en mí cuando más los necesité.

A todas las personas que formaron parte de mi vida durante estos cinco años en especial a Yannier por enseñarme la vida desde otra perspectiva.

A aquellas personitas que conocí en esta universidad y que tendrán siempre un lugar en mi corazón: Lilianet, Yanet, Dariannys, Daymaris, Miguel, Pedro Pablo, Johan y Peña entre otros, gracias a todos.

A todos lo que de una forma u otra contribuyeron a que mi sueño se hiciera realidad.

Dedicatoria.

Este trabajo va dedicado a las personas más importantes de mi vida:

A mis padres por ser mis mayores amigos, mis cómplices, las personas que me alentaron a seguir cuando ya no tenía fuerzas, por ser los mejores padres de este mundo.

A mis hermanos especialmente a Leydis, para que sea una fuente de inspiración a seguir adelante.

Resumen.

En el año 2012 el departamento de Sistemas Integrales en Migración, Asesoría y Soporte perteneciente al Centro de Software Libre de la Universidad de las Ciencias Informáticas, desarrolló una Herramienta para Migrar y Administrar Servicios Telemáticos en los Organismos de la Administración Central del Estado. Desarrollar un módulo para monitorizar la integridad de los archivos en los servidores que se administran desde dicha herramienta es el objetivo de la presente investigación con el fin de elevar la seguridad en los mismos.

Para esto se realizó un estudio de las herramientas que permiten monitorizar la integridad de los ficheros entre las cuales se destacó AFICK por cumplir el mayor número de funcionalidades requeridas por el cliente. Además, se documentan las tecnologías, herramientas y lenguajes de programación utilizados, así como los artefactos requeridos por la metodología de desarrollo utilizada: SXP.

El módulo desarrollado cuenta con funcionalidades que permiten instalar y desinstalar el software, gestionar sus ficheros de configuración y de base de datos, administrar la realización de los chequeos y mostrar los reportes de los mismos.

Palabras claves: AFICK, integridad de ficheros, monitorización, seguridad informática.

Índice de Contenido.

Índice de Figuras.

Introducción.

Desde los inicios de su desarrollo el hombre se preocupó por preservar los objetos que le pertenecían con el objetivo de darle seguridad ante cualquier circunstancia. Con la llegada de las computadoras a sus vidas y con ellas la computación, el advenimiento de la era digital y el Internet, las personas se vieron en la necesidad de proteger también aquella información que contenían sus ordenadores, así surgió lo que se conoce hoy como la seguridad informática.

La seguridad informática es un campo relativamente amplio, que puede traducirse de diferentes formas de acuerdo con el radio de acción del medio en que se aplique. A lo largo de los años ha ido cobrando fuerzas y potenciando su definición luego de ser analizada por los expertos de las ciencias de la computación y la informática.

Un sistema que gestiona la seguridad informática debe mantener la confiabilidad, la disponibilidad y la integridad. Para la protección de los sistemas de cómputo se debe realizar un análisis de las amenazas potenciales que pueden sufrir los mismos, las pérdidas que podrían generar, y la probabilidad de su ocurrencia. A partir de ese análisis se diseña una política de seguridad que defina las responsabilidades y reglas a seguir para evitar tales amenazas o minimizar sus efectos en caso de que se produzcan. A las herramientas utilizadas para implementar esta política de seguridad se les denomina mecanismos de seguridad, los cuales son la parte más visible del sistema de seguridad que garantiza la protección de los sistemas de cómputo.

Generalmente, la taxonomía más elemental de las amenazas se divide en tres grandes grupos: interrupción, intercepción y modificación, por lo que es de vital importancia la protección de los elementos fundamentales de todo sistema de cómputo: el software, el hardware y los datos [1]. Para garantizar la seguridad del sistema, es necesario prevenir, evitar y detectar intrusiones, siendo la detección de intrusos un elemento que se debe tener en cuenta a un nivel más alto pues hay que tomar las medidas necesarias para que el sistema no colapse totalmente una vez franqueados los mecanismos de defensa.

Existen herramientas para proteger los archivos del sistema donde se encuentren, este es el objetivo de los verificadores de integridad que a través de la supervisión constante de los cambios en ciertos archivos y carpetas en intervalos regulares de tiempo, son capaces de indicar si hubo alguna modificación y/o si está en peligro. Esto es debido a que los datos constituyen el principal elemento a proteger por ser el activo más amenazado.

En realidad la seguridad de un sistema depende de la empresa, organización o utilización que tenga, aunque en cualquier entorno Unix se ofrecen los mecanismos de seguridad suficientes para satisfacer las necesidades de la mayoría de las instituciones. En este sistema se han añadido numerosas herramientas para alcanzar mayores niveles de seguridad, se puede considerar el sistema operativo de propósito general más fiable del mercado [2].

GNU/Linux es un sistema operativo basado en Unix que es 100% Software Libre en el cual los usuarios tienen la libertad de copiar, distribuir, estudiar, modificar y mejorar el software [3]. Este modelo de desarrollo, que siguen tanto GNU/Linux como la mayoría de las aplicaciones que se ejecutan sobre él, conduce a altos niveles de seguridad. Pues es real que cualquier usuario que cuente con los conocimientos requeridos puede acceder al código fuente para encontrar las debilidades, pero gracias a la comunidad que sustenta el desarrollo de GNU/Linux y sus aplicaciones, el tiempo que demora en aparecer la corrección de cualquier vulnerabilidad o debilidad en el código fuente es relativamente corto en comparación con otros sistemas operativos, donde este proceso puede demorar semanas o incluso meses. Debido a esto es conocido por su alto nivel de estabilidad, que parte del propio núcleo del sistema operativo lo que propiamente es Linux.

Cuba se encuentra inmersa en el proceso de migración a Aplicaciones de Código Abierto, en el cual tiene un papel protagónico la Universidad de las Ciencias Informáticas (UCI). El Centro de Software Libre (CESOL) de la UCI, al cual pertenece el Departamento de Servicios Integrales en Migración, Asesoría y Soporte (SIMAYS), cuenta con una Herramienta para la Migración y Administración de Servicios Telemáticos (HMAST), la cual fue creada para ser utilizada en el proceso de migración de los Organismos de la Administración Central del Estado (OACE). Además, dicha herramienta tiene por objetivo velar por la disponibilidad de los servicios telemáticos que se utilizan en los OACE haciéndose necesario un aumento de la seguridad en los servidores que brindan los citados servicios.

Actualmente HMAST posee módulos para la administración de los servicios SSH y DHCP, sin embargo, no existe un mecanismo que permita configurar las políticas de seguridad necesarias para detectar alteraciones que comprometan la integridad de los archivos en los servidores que se administran con esta herramienta. Dada esta problemática se hace necesaria la implementación de un mecanismo que se encargue de monitorizar la integridad de los archivos de manera autónoma.

Por lo que se plantea el **problema científico** siguiente: ¿Cómo monitorizar la integridad de los archivos en los servidores administrados con HMAST?

Identificándose como **objeto de estudio** los mecanismos de seguridad de código abierto para la monitorización de la integridad de los ficheros, por lo que el **campo de acción** está centrado en la monitorización de la integridad de los archivos desde la herramienta HMAST.

Como **objetivo general** se ha planteado desarrollar un módulo para HMAST que permita monitorizar la integridad de los archivos en los servidores. Desglosándose en los siguientes **objetivos específicos**:

- Sistematizar los conceptos relacionados con la integridad de los archivos.
- Diseñar el módulo para monitorizar la integridad de los archivos.
- Implementar el módulo para monitorizar la integridad de los archivos.
- Probar el módulo implementado.

Para darle cumplimiento a los objetivos anteriormente expuestos se definen una serie de **tareas de investigación**:

- Análisis de la documentación relacionada con la seguridad informática en cuanto a la integridad de los ficheros en escenarios de código abierto.
- Realización de un estudio acerca de los sistemas de verificación de integridad de los archivos.
- Documentación de los requisitos del módulo para monitorizar la integridad de los archivos.
- Descripción de la arquitectura del módulo para monitorizar la integridad de los archivos.
- Creación del código que conforma la propuesta de solución del problema.
- Realización de pruebas al módulo para monitorizar la integridad de los archivos.

Se define la siguiente **idea a defender:** el desarrollo del módulo para monitorizar la integridad de los archivos desde HMAST brindará la posibilidad de elevar los niveles de seguridad en los servidores administrados.

Para llevar a buen término el trabajo se utilizaron varios métodos de investigación científica, siendo estos:

Métodos teóricos:

Análisis y síntesis: se utilizó principalmente en la precisión de los fundamentos teóricos relacionados con la seguridad informática en cuanto a la integridad de los archivos, para analizar, valorar y conocer las particularidades del proceso de protección de los archivos y las herramientas empleadas para ello.

Inducción y deducción: este método permitió analizar el fenómeno de la integridad de los archivos partiendo de las características propias de la seguridad informática, el proceso de la administración de las políticas de seguridad y la influencia que tiene en otros contextos para arribar a conclusiones de acuerdo con lo investigado.

Estructura del contenido y breve explicación de sus partes.

La presente investigación comprende la introducción, tres capítulos, las conclusiones generales, las referencias bibliográficas y los anexos. La estructura de los capítulos se define a continuación:

Capítulo 1. Fundamentación teórica. Se expone el marco conceptual de la propuesta de solución, introduciendo conceptos e ideas que se manejarán a lo largo de la investigación. Comprende el estudio del estado del arte de las herramientas que monitorizan la integridad de los archivos. Además, se caracterizan las herramientas, la metodología y los lenguajes de programación que serán utilizados en el desarrollo de la solución.

Capítulo 2. Características del sistema. Enmarca la descripción de la solución propuesta. En un primer momento se enuncian las características del sistema para luego modelar el mismo haciendo uso de la metodología de desarrollo de software y se presentan los principales artefactos generados en las fases que esta propone.

Capítulo 3. Implementación y pruebas del sistema. Comprende la implementación del sistema, las Tareas de Ingeniería de software que complementaron la misma y finalmente la evaluación de la solución a través de los resultados obtenidos una vez realizadas las pruebas al sistema.

Capítulo 1: Fundamentación teórica.

En este capítulo se desarrollan dos puntos fundamentales, primeramente la descripción de los principales conceptos teóricos vinculados con el desarrollo de la investigación para facilitar el entendimiento a partir de distintos enfoques, definiciones y puntos de vistas.

En un segundo momento prevalece la descripción de las herramientas, la metodología y los lenguajes de programación que se utilizaron en el desarrollo de la solución propuesta.

1.1 Principales conceptos.

A continuación se abordan los conceptos asociados a la presente investigación, haciendo énfasis en los principales aspectos a tener en cuenta en el desarrollo de la misma.

1.1.1 Seguridad Informática.

La seguridad informática es una disciplina que se encarga de proteger los sistemas de cómputo tratando de asegurar la información contenida en estos. Aunque mayormente cuando se usa el término se piensa muy rápido en la seguridad de la información que no es más que la preservación de los datos sin tener en cuenta el medio donde se localice: ya sea impreso en papel, o en un disco duro [4]. La seguridad informática va mucho más allá, fundamentalmente se encarga de preservar:

- La confidencialidad que consiste en mantener la privacidad de la información en el sistema a partir de las reglas que se establezcan para el acceso a la misma. La información no debe ser revelada ni descubierta más que a las personas autorizadas y en el tiempo y forma que se haya convenido.
- La integridad o nivel de actualización, autenticidad y completamiento de la información. Es la propiedad por la cual se mide que la información sea exacta y completa, que no sea alterada más que por las personas autorizadas.
- La disponibilidad que garantiza el acceso a la información en cualquier momento, para usuarios autorizados.

Los sistemas pueden ser protegidos desde un punto de vista lógico y físico. El activo más importante que se posee en un sistema de cómputo es la información y, por lo tanto, deben existir

técnicas que la aseguren, más allá de la seguridad física que se establezca sobre los equipos en los cuales se almacena. Estas técnicas las brinda la seguridad lógica que consiste en la aplicación de barreras y procedimientos que resguardan el acceso a los datos y solo permiten acceder a ellos a las personas autorizadas para hacerlo. La presente investigación se enmarca a uno de los grupos lógicos que define la seguridad informática: la integridad.

1.1.2 Integridad de los archivos.

Comprobar la integridad de un fichero consiste en averiguar si algún dato del archivo ha variado desde su creación. El archivo puede haber sido modificado por error, por cortes en la comunicación o en el peor de los casos porque un atacante haya inyectado[1] código malicioso.

En entornos de código abierto es posible contrastar y compilar el código fuente del programa, revisándolo para comprobar que no contiene código no deseado. De esta forma, se puede garantizar que el software no ocasionará ningún daño, aunque por el volumen de algunos programas (y por tanto de su código fuente), no es un proceso práctico en la realidad [5].

Muchas instituciones han definido la integridad de los archivos desde diferentes ópticas según sus puntos de vistas y adoptando posiciones particulares, por lo que, no es objetivo restringir este debatido concepto a una definición, sino que se persigue lograr un entendimiento más amplio que permita conformar juicios propios y acertados, considerando una variedad de perspectivas que tal vez sean mejor asimiladas ya sea porque posean mayor nivel de sencillez, generalidad, especificidad, etc.

Para lograr este objetivo, se hace referencia a las siguientes definiciones sobre la integridad de los archivos, una de ellas, definida según la norma ISO 27001, es mantener la exactitud y completitud de la información y de los métodos de su proceso [6].

Por otra parte, en el documento de Gestión de Riesgos en la Seguridad Informática queda plasmado sobre la integridad que los datos son completos, non-modificados y todos los cambios son reproducibles, se conoce el autor y el momento del cambio [7].

1 Significa en términos de programación, inserción de código.

La presente investigación se acoge fundamentalmente a la dada por el Instituto Nacional de Tecnologías de la Comunicación (INTECO) de España el cual define que la integridad consiste en que nadie puede cambiar, recortar o falsificar ilegítimamente los datos [8].

1.2 Herramientas de verificación de integridad de ficheros.

Los verificadores de integridad son un subconjunto de los Sistemas de Detección de Intrusos o Intrusiones (SDI) para estaciones de trabajo. Básicamente estos programas se ocupan de comprobar la integridad de los ficheros del sistema donde se ubican. Ya es tradicional que este trabajo se realice empleando sumas de verificación que son ejecutadas cada cierto intervalo de tiempo y sobre los ficheros seleccionados por el administrador del sistema. Dichos programas suelen verificar los permisos en archivos y directorios, las cuentas de usuarios y sentencias en el registro del cron[2] de Unix. De manera general no emiten alarmas, como el resto de los SDI, sino que generan registros (logs) con los resultados del trabajo que realizan [9].

1.2.1 Tripwire.

Es una aplicación de código abierto para sistemas operativos GNU/Linux, aunque existe una versión comercial para Windows, su funcionamiento consiste en avisar al administrador de cualquier cambio o alteración en los archivos de la máquina, incluidos los binarios. El programa crea una base de datos con un identificador por cada archivo analizado y puede comparar en cualquier momento el actual con el registrado en la base de datos, avisando ante cualquier alteración, eliminación o inclusión de un archivo en el sistema [10].

Tripwire, permite configurar las reglas o políticas por las que se chequearán los ficheros y directorios. Todas las reglas se definen en el archivo de configuración de la herramienta: twconfig.txt, donde se describen las condiciones en que Tripwire debe notificar.

El mensaje de alerta es una de las características más importantes de Tripwire. Este enviará un reporte con el resultado del chequeo realizado por correo electrónico a las direcciones que se le hayan configurado, tan pronto como se detecte una intrusión, lo cual permitirá al administrador tomar acciones que eviten una pérdida de información o un colapso en el sistema.

2 Administrador de procesos en segundo plano a intervalos regulares de tiempo.

No es un demonio[3], por lo que no opera en tiempo real, aunque permite configuraciones para auto ejecutarse cada cierto tiempo. Por cada verificación realizada guarda un informe donde se podrán ver los cambios que se han realizado en el sistema que se está supervisando.

El archivo de base de datos mencionado anteriormente se crea inmediatamente después de instalar Tripwire. Esta base de datos puede ser modificada, sin embargo, solo se debe crear una vez. Es decir, si ya se ha inicializado la base de datos, para darse cuenta de que algo no se ha configurado correctamente no se debe eliminar el archivo sino que se deben realizar cambios en la configuración y aplicarlos a la base de datos [11].

Tripwire codifica todos sus archivos con una firma de 1024 bits. Esto se hace mediante el uso de un conjunto de pares de claves, con una pública y una privada. Una de ellas es la clave del sitio, que se utiliza para proteger el orden y los archivos de configuración de un sitio. La otra es la clave local, que se utiliza para proteger el archivo de base de datos y los archivos de informe que se tenga. Con la clave de sitio, un administrador del sistema puede desarrollar un archivo de política único para todo un sitio. Ese archivo de política sería accesible con una contraseña que el administrador conocería.

Tripwire aunque es una candidata excelente no es la más conveniente a utilizar para monitorizar la integridad de los archivos desde HMAST, pues aunque es muy completa y sus características son muy buenas, la lectura y escritura de su fichero de configuración no es sencilla. Se precisa de un conocimiento fundamentado por un buen estudio de la herramienta, lo cual es un proceso bastante extenso. Su sintaxis representa todo un lenguaje de la propia aplicación. Por lo que su configuración automática requiere la implementación de un intérprete de su lenguaje de políticas, que no es algo trivial.

1.2.2 iWatch.

Es una herramienta de monitorización de ficheros en tiempo real para sistemas GNU/Linux, puede ejecutarse en dos modos [12]:

- **Como demonio:** donde se ejecuta como demonio del sistema y supervisa los archivos indicados en el fichero de configuración iwatch.xml.

3 Programa o aplicación que se ejecuta en tiempo real.

- **En línea de comandos:** donde se le especifican todos los parámetros, blanco a monitorizar, dirección de correo a la que enviar informe, excepciones, etc. Este método es la elección más adecuada para tareas de monitorización más complejas.

Esta aplicación puede ser muy útil para monitorizar archivos sensibles o directorios frente a cualquier cambio, como por ejemplo monitorizar los archivos ubicados en "/etc/password" o supervisar el directorio raíz contra cualquier cambio indeseado.

En el archivo de configuración cada grupo de reglas puede tener su propio punto de contacto de correo electrónico. Este contacto recibirá una notificación por correo electrónico de cualquier cambio en los objetivos monitorizados. Puede supervisar un directorio recursivamente, y también configurar una lista de excepciones, con los directorios o archivos dentro de otros directorios monitoreados que no se deseen supervisar. También es posible desactivar la notificación de correo electrónico, y en lugar de enviar un correo configurar un comando a ser ejecutado si se produce un evento.

Es una herramienta muy práctica que hace accesible la interfaz Inotify[4] al usuario. En lugar de usar la API[5] del Kernel para monitorizar archivos, solo se necesita un archivo de formato XML muy intuitivo para la monitorización de áreas específicas del sistema de archivos. Dado que el script Perl de la herramienta está bien diseñado resulta fácil de leer y sencillo de personalizar [13].

La herramienta **iWatch** no es la más efectiva a utilizar para monitorizar la integridad de los archivos desde HMAST pues sus características son muy básicas para custodiar los ficheros en servidores. Además, su ejecución ideal para ficheros muy sensibles es a modo de demonio, lo que representa una dificultad, pues con solo detener la aplicación todo el sistema está totalmente desprotegido ante cambios en sus ficheros.

1.2.3 Integrit.

4 Inotify es un subsistema del kernel Linux que actúa para extender los sistemas de ficheros a notar cambios en el sistema de archivos, e informar de los cambios a las aplicaciones.

5 Interfaz de Programación de Aplicaciones, API por sus siglas en inglés.

Es una herramienta de código abierto que ayuda a determinar si han ocurrido modificaciones en el sistema. Con el uso de Integrit un administrador de software conocería si los programas que usa para investigar el sistema son troyanos o no pues esta permite realizar búsqueda de malware[6] en el mismo. La herramienta funciona creando una base de datos que es una imagen de las partes esenciales del sistema, para después usarla en asegurarse de que nadie ha realizado ninguna modificación ilícita en su sistema de archivos.

Entre las características de Integrit se puede destacar el bajo consumo de memoria, el conjunto de reglas en cascada intuitiva para los listados de rutas en el archivo de configuración, la posibilidad de obtener una salida con el resultado de su trabajo en formato XML o en texto plano, y las comprobaciones y actualizaciones simultáneas [14]. Tiene características básicas como incluir la suma de control MD5 de bases de datos recién generadas en el informe.

Al crear una base de datos en modo de actualización, que es una instantánea de un sistema host[7] en un estado conocido, los archivos del host más tarde pueden ser verificados como inalterados ejecutando Integrit en el modo de verificación para comparar el estado actual con el estado grabado conocido.

Por otra parte **Integrit** no es la herramienta seleccionada para monitorizar la integridad de los archivos desde HMAST, pues para ello se le debe especificar un directorio raíz, si se desea chequear directorios diferentes se debe tener otro archivo de configuración lo que trae consigo que se tengan varios de estos, lo que resultaría engorroso a la hora de trabajar con varios servidores. Además la herramienta se adentra en conceptos que no son de interés para la investigación como es la búsqueda de virus.

1.2.4 AFICK.

Another File Integrity Checker (AFICK) es una herramienta que permite monitorizar los cambios en los sistemas de archivos, por lo que puede detectar intrusiones. Está diseñada para ser rápida y portátil, funciona en los sistemas operativos Windows con Active Perl y en diferentes distribuciones de GNU/Linux, pero debe funcionar en cualquier ordenador con el lenguaje de

6 Programas malignos.

7 El término host es usado en informática para referirse a las computadoras conectadas a una red, que proveen y utilizan servicios de ella.

scripting[8] Perl y sus módulos estándar [15]. El proceso de monitorización consiste en crear una base de datos con una copia de los ficheros a custodiar y luego aplicar una serie de reglas, definidas en su fichero de configuración para comprobar el estado de los mismos.

La herramienta maneja términos para englobar secciones que facilitan el modo en que realiza su trabajo. Una de estas secciones es la de las directivas en la cual se configuran varios parámetros para indicar cuán profundo y a qué nivel de completitud y exactitud se monitorizarán los ficheros. Otra sección no menos importante es la de las macros, esta permite configurar si se automatizará el proceso o no, ya que la herramienta no se ejecuta en tiempo real.

Una de las características más importantes que posee es el mensaje de alerta. AFICK puede ejecutarse automáticamente cada cierto tiempo a través del cron de Unix. El resultado de dicho trabajo genera un reporte el cual se envía por correo electrónico si el administrador lo desea. Puede enviarse uno por cada chequeo o solamente en caso que existan modificaciones en los archivos monitorizados. Una respuesta rápida permitirá encontrar una posible intrusión o evitar que un ataque sea exitoso. Aunque se debe modificar en el archivo de configuración los archivos que generen una cantidad de alertas considerable dado que son modificados constantemente. Una vez que los reportes sean estándar se podrá configurar el tiempo de chequeo automático y el envío de correos electrónicos. Es una versión un poco más sencilla de Tripwire, puede decirse así ya que tienen varias similitudes, aunque AFICK resulta más fácil de configurar [16].

La autora considera que la herramienta **AFICK** es la más efectiva a utilizar para monitorizar la integridad de los archivos desde HMAST. La lectura y escritura de su fichero de configuración es sencilla, por lo que resulta fácil de configurar automáticamente. Además, permite realizar configuraciones generales, evitando un volumen redundante de reglas o políticas.

Al no encriptar el fichero de configuración es más sencilla de administrar, no obstante logra un nivel de seguridad aceptable guardando sus configuraciones y base de datos con permisos de lectura y escritura solo para los administradores.

1.3 Comparativa entre las herramientas de verificación de integridad.

8 Un lenguaje de scripting es un lenguaje de programación que puede manipular la funcionalidad de un sistema informático configurado para proporcionar el intérprete de este lenguaje y entorno de una interfaz que determinan las posibilidades de la misma.

Los sistemas operativos de tipo Unix emplean una estructura propia para los sistemas de archivos, la cual contiene características de los archivos, directorios y otros objetos del sistema de ficheros. El estándar POSIX[9] establece un modelo de sistema de archivos que se ajusta al empleado en Unix, debido a esto un archivo tendrá las siguientes propiedades: identificador, número de nodo índice, longitud del archivo, identificador de usuario y grupo, modo de acceso, marcas de tiempo y número de enlaces. Para verificar la integridad de un fichero es necesario comprobar que no haya sido sometido a cambios, para esto se evalúa que sus propiedades no varíen de forma inesperada sino que permanezcan tal y como el administrador espera [17].

Seguidamente se muestra una pequeña comparación sobre cómo las herramientas analizadas anteriormente se manifiestan con respecto al análisis de las características mencionadas y otras no menos importantes a tener en cuenta en la propuesta de solución.

Características	**AFICK**	**iWatch**	**Tripwire**	**Integrity**
Fácil de configurar	x	x		x
Emplea archivo base de datos	x		x	x
Verifica Md5	x	x	x	x
Verifica Sha1	x	x	x	
Verifica el número de nodo índice	x	x	x	x
Verifica el grupo del fichero	x		x	x
Verifica el tamaño del fichero	x		x	x
Verifica fecha de creado	x	x	x	x
Verifica el número de bloques destinados al fichero	x		x	
Verifica el número de enlaces	x		x	
Verifica el tiempo de la última modificación del contenido del fichero	x	x	x	x
Verifica el tiempo del último cambio en las propiedades del fichero	x		x	x
Verifica el usuario propietario del fichero	x	x	x	x

9 Interfaz de Sistemas Operativos Portables, POSIX por sus siglas en inglés.

Verifica el tiempo del último acceso	x		x	x
Verifica los permisos del fichero	x	x	x	x
Genera reportes	x		x	x
Uso de crontab	x	x	x	
Uso de syslog para guardar los registros	x	x	x	x
Notificación vía correo electrónico	x	x	x	

Tabla 1: Comparativa entre las herramientas de verificación

Una vez analizadas las principales características de las herramientas de verificación de integridad y realizada la comparación entre ellas que llevó a solidificar la selección de AFICK para el módulo de integridad de ficheros de HMAST se caracteriza la metodología que guiará el desarrollo de la propuesta de solución. La cual está definida en el documento de “Concepción del sistema HMAST” en el expediente de proyecto de la citada herramienta.

1.4 Metodología de desarrollo de software.

Las metodologías de desarrollo de software son un conjunto de procedimientos y técnicas que ayudan a la documentación para el desarrollo de productos software [18]. Existen diferentes tipos como son las ágiles, las pesadas o tradicionales, en espiral, entre otras. Cada una dirigida al modo de desarrollar y a las características de cada equipo de desarrollo.

Las metodologías ágiles intentan evitar los engorrosos caminos usados en las metodologías tradicionales, enfocándose en las personas y los resultados. Se basan en promover iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto, logrando que se minimicen los riesgos desarrollando software en cortos lapsos de tiempo.

El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar poco tiempo. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener una versión al final de cada iteración, donde el equipo vuelve a evaluar las prioridades del proyecto.

Por las grandes ventajas que proporcionan estas metodologías se propone para el desarrollo de esta investigación el uso de la metodología ágil SXP. Además, está orientado por la dirección del centro el uso de esta en los trabajos que pertenezcan a él [19].

1.4.1 Metodología ágil de desarrollo SXP.

SXP es una metodología de desarrollo de software compuesta por las metodologías SCRUM y XP que ofrece una estrategia tecnológica a partir de la introducción de procedimientos ágiles que permitan actualizar los procesos de software para el mejoramiento de la actividad productiva fomentando el desarrollo de la creatividad, aumentando el nivel de preocupación, responsabilidad de los miembros del equipo y ayudando al líder del proyecto a tener un mejor control del mismo.

SXP está especialmente indicada para proyectos de pequeños equipos de trabajo y rápidos cambios de requisitos, donde existe un alto riesgo técnico y se orienta a una entrega rápida de resultados y una alta flexibilidad. Ayuda a que trabajen todos juntos, en la misma dirección, con un objetivo claro, permitiendo además seguir de forma clara el avance de las tareas a realizar, de tal manera que se pueda palpar día a día cómo progresa el trabajo [20].

1.5 Tecnologías a utilizar.

En el presente epígrafe se describen las herramientas a utilizar para el desarrollo del sistema. La selección de las mismas se corresponde a la realizada en el documento de “Concepción del sistema HMAST” en el expediente de proyecto de la citada herramienta.

1.5.1 Visual Paradigm Suite.

Visual Paradigm es una herramienta para el modelado UML[10] profesional que soporta el ciclo de vida completo del desarrollo de software*:* análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.

10 Lenguaje de Modelación Unificado, UML por sus siglas en inglés.

Esta herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas y proyectos [21].

La misma fue utilizada en la presente investigación para la realización de los diagramas de clases y paquetes, para la representación del modelo de dominio y para el diagrama de componentes. Además, para generar el código inicial de la propuesta de solución a partir del diagrama de clases y paquetes.

1.5.2 RapidSVN.

RapidSVN es un cliente de interfaz gráfica para la comunicación con servidores Subversion. Este facilita el mantenimiento de las diferentes versiones de ficheros desde una interfaz sencilla e intuitiva. Es una herramienta rápida y eficiente. Es fácil de usar, tanto por quienes ya conocen Subversion como para quien empieza, pudiendo acceder a direcciones SVN, subir y descargar contenido y sincronizarlo con el servidor original, comprobar su estado, crear y fusionar direcciones [22].

El uso de esta herramienta está fundamentado en el control de versiones de la documentación de la investigación y el código fuente relacionado a la solución del problema planteado.

1.5.3 NetBeans.

El IDE NetBeans es un producto libre y gratuito sin restricciones de uso, es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Se puede ejecutar en los sistemas operativos Windows, Mac OS, Linux, Solaris y otros. Existe además un número importante de módulos para extender el uso de NetBeans, lo que posibilita que el mismo pueda dar soporte para otros lenguajes de programación [23].

1.5.4 Framework Spring.

En el desarrollo de software, Spring es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java [24]. Por su diseño, ofrece mucha libertad a los desarrolladores en Java, así como soluciones bien documentadas y fáciles de usar para las prácticas comunes. Fundamentalmente se compone de:

- Contenedor de inversión de control (IoC).

- Aplicación de la técnica de inyección de dependencias.
- Un framework AOP[11].
- Una capa de abstracción de servicios.
- Integración consistente con varios estándares y APIs populares.

En el desarrollo de la solución es el encargado de leer los archivos XML con la información de la configuración de cada módulo en cada servidor así como también las clases que contienen los beans necesarios para acceder a las clases en la capa de aplicación.

1.6 Lenguajes utilizados.

Un lenguaje de programación consiste en un conjunto de reglas sintácticas y semánticas que definen un lenguaje informático [25]. Permite al programador especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o trasmitidos y qué acciones debe tomar bajo una variada gama de circunstancias [26].

A continuación se exponen las principales característica de los lenguajes de programación que serán utilizados en el desarrollo del módulo, los que se corresponden con los seleccionados en la concepción del sistema base definidos en el expediente de proyecto del mismo.

Java.

Este lenguaje cuenta con una interfaz orientada a objetos para acceder de un modo portable a cualquier base de datos, promoviendo la portabilidad. Es poseedor de una extensa biblioteca utilitaria y la portabilidad alcanzada es cualitativamente superior a la que se puede obtener con los lenguajes C/C++ [27].

Actualmente, dentro de los lenguajes populares, es uno de los mejores en cuanto a definición debido a que tiene total independencia del implementador del lenguaje y de sus clases auxiliares. Proporciona los tipos de datos primitivos similares a los de C++, solo que carece de punteros y mediante su biblioteca de clases estándar proporciona todas las estructuras contenedoras clásicas. Tiene cuatro niveles de empaquetamiento: variables y funciones, al igual que C++ y otros dos propios de él, denominados: clases y paquetes [28].

11 Programación orientada a Aspectos, AOP por sus siglas en inglés.

Es utilizado el lenguaje para generación de código del lado del servidor, a través de este se crearon las clases e interfaces donde está contenido el código que recoge toda la lógica del sistema.

XML.

Es un lenguaje de marcas generalizado utilizado para estructurar la información en cualquier documento que contenga texto (por ejemplo: archivos de configuración de un programa en particular o una base de datos) [29]. En la presente investigación es utilizado para la configuración de Spring y la estructura de las configuraciones de los servidores y el fichero de configuración de AFICK.

Conclusiones parciales.

A lo largo de la investigación realizada se pudo adquirir la base teórica necesaria para poder lograr los objetivos trazados en el desarrollo de la solución propuesta. Para ello, se abordaron los principales conceptos relacionados con la temática a tratar.

Partiendo de la necesidad que posee HMAST de contar con un módulo que le permita monitorizar la integridad de los ficheros en los servidores que administra, se realizó un estudio de las herramientas que realizan dicho proceso. El objetivo principal de este estudio fue seleccionar una herramienta que permita lograr la meta propuesta lo cual fue posible.

Además, se describieron las tecnologías y la metodología de desarrollo de software que se utilizarán durante el ciclo de desarrollo de la propuesta de solución.

Capítulo 2: Características del sistema.

En este capítulo se exponen las funcionalidades que debe cumplir el sistema, las cuales están descritas mediante las Historias de Usuario. Además, se describe la arquitectura y el diseño que tiene el mismo.

2.1 Descripción del sistema base: HMAST.

HMAST es una herramienta que permite administrar los servicios telemáticos de forma remota, la cual tiene las funcionalidades necesarias para administrar los usuarios, las tareas programadas y los servicios, entre otras. La arquitectura que presenta la herramienta HMAST propone el diseño de una arquitectura N-Capas orientada al dominio y está compuesta por 5 capas (Véase Figura 1) que se describen a continuación:

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** que es responsable de promover la reutilización de código, ella albergará las operaciones de seguridad, registro, monitoreo del sistema, mecanismos de persistencia reutilizables, validadores genéricos, en fin todas aquellas operaciones que se puedan llamar desde otras capas.

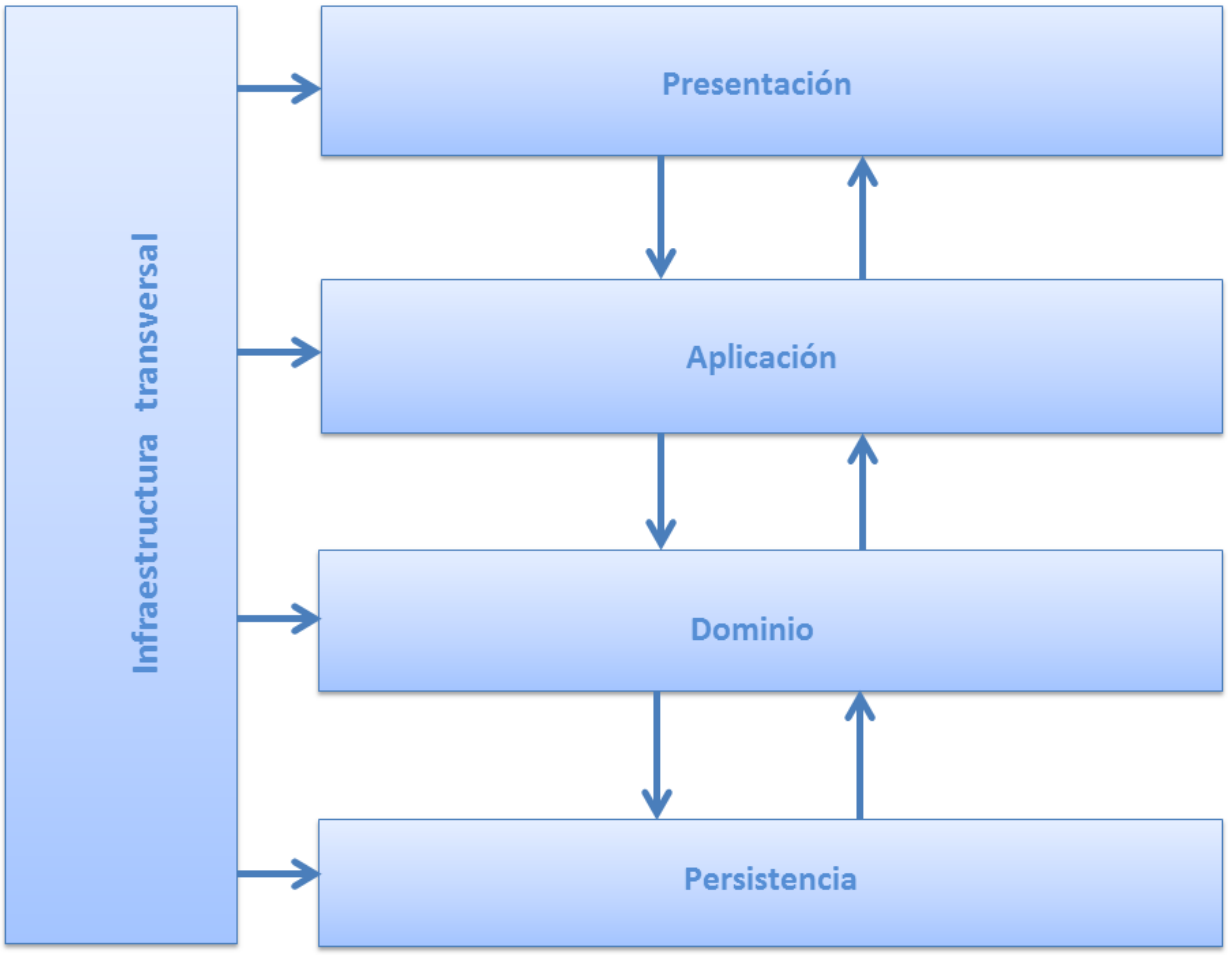


Figura 1: Arquitectura de HMAST

2.1.1 Funcionalidades que brinda HMAST.

- Gestión de servidores lógicos: Permite la adición, edición, eliminación de los datos de un servidor lógico, además permite la conexión remota y desconexión a un servidor seleccionado.

- Gestión de servicios telemáticos asociados a un servidor lógico: Permite la adición, edición, eliminación de los datos de un módulo, así como activación y desactivación de los mismos.

- Gestión de las variables de configuración asociadas a un servidor lógico: Permite cargar y salvar las variables de configuración de los servicios telemáticos instalados en un servidor lógico (ficheros de configuración, nombre de módulos, demonios, etc).

2.1.2 Consideraciones para implementar un módulo para HMAST.

- La lógica de Aplicación no deberá incluir ninguna lógica del Dominio, solo tareas de coordinación relativas a requerimientos técnicos de la aplicación, como conversiones de formatos de datos de entrada a entidades del Dominio y llamadas a componentes de Infraestructura para que realicen tareas complementarias.

- Se debe garantizar que no viajen hacia y desde la capa de presentación objetos de dominio, en su lugar deben viajar objetos DTO (*Data Object Transfer*).

- Las entidades solo pueden tener dependencias de componentes de la capa de dominio.

- 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.

Una vez descrito el sistema base en el que se encontrará el módulo para monitorizar la integridad de los archivos (MMIAS) se detallan las características de dicha propuesta la cual está condicionada a las características del sistema base a petición del cliente por cuestiones organizativas.

2.2 Conceptos asociados al dominio del problema.

En el ámbito asociado al dominio de la investigación, se identificó que no se manejan los procesos de negocio. El motivo es que resulta difícil captar claramente estos procesos, imposibilitando determinar el conjunto estructurado de las actividades que se desarrollan en el negocio, toda la organización, las relaciones, etc. Es por ello que se determinó el uso del modelo de dominio para comprender los conceptos con los que se trabajan en el desarrollo y puesta en práctica de la aplicación. Básicamente, el modelo de dominio se representa como uno o un

conjunto de diagramas de clases y engloba los conceptos de la propia realidad física en la que ejerce el sistema [30].

Con la construcción del modelo de dominio se persigue plasmar el conocimiento adquirido. Es utilizado como una útil herramienta para comprender el sector al cual el sistema va a servir. Una vez completado este paso es probable que se desarrolle una amplia visión del sistema y el entorno en que este será desplegado posteriormente.

Concretamente, ¿qué elementos quedarán representados en el diagrama de dominio?, a continuación se listan los que se consideran más relevantes [31]:

- Conceptos u objetos del dominio del problema: clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

Esto no quiere decir que representar cada uno de los elementos antes mencionados sea un procedimiento estricto. Es posible capturar un mayor grado de detalle en dependencia de lo que se quiera lograr. Por esta razón está abierta la decisión de cuanto detalle va a ser necesario o no incluir en cada modelo. El objetivo es captar lo suficiente para entender el medio en el cual el sistema que se está diseñando va a funcionar y esto demanda una cantidad distinta de detalles cada vez.

A continuación se muestra el modelo de dominio del módulo de HMAST para monitorizar la integridad de los archivos.

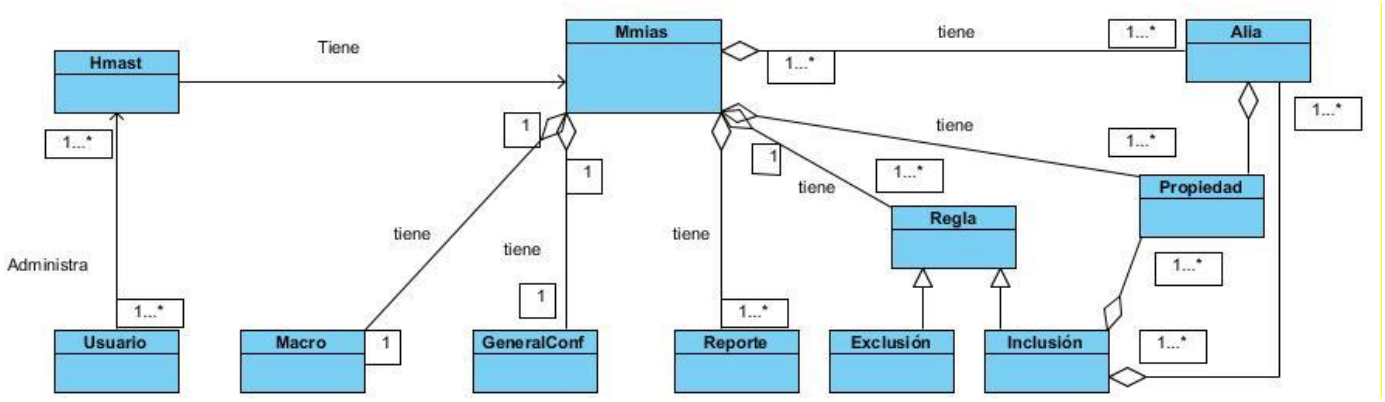


Figura 2: Modelo de dominio de MMIAS

Como se muestra en la figura anterior un usuario administra HMAST que a su vez contará con el módulo MMIAS, el mismo estará compuesto por las macros, las configuraciones generales, una lista de reglas: las cuales pueden ser de inclusión y de exclusión de ficheros, directorios o de propiedades, una lista de alias y una lista de reportes. Seguidamente se detallan dichas clases:

- **Mmias:** representa el sistema, el cual contará con las configuraciones generales (Config-Gen) de la herramienta, las macros, los alias, un grupo de reportes y un conjunto de reglas.
- **ConfigGen:** representa las configuraciones del sistema, donde se recogen todas las variables influyentes en el "cómo" sobre el funcionamiento de la herramienta.
- **Macro:** representa las macros donde se encuentran los valores para la configuración de AFICK cuando se ejecuta a través del cron de Unix.
- **Reporte:** representa el conjunto de registros generados por la herramienta durante su funcionamiento.
- **Regla:** representa las sentencias por las cuales se realiza un chequeo de integridad, las reglas de inclusión están compuestas por la ruta de un fichero o directorio y las propiedades que se desean verificar del mismo; por su parte las de exclusión por un símbolo y una ruta.
- **Alia:** representa un conjunto de propiedades englobadas o representadas por un nombre.
- **Propiedad:** representa todas las propiedades que se pueden chequear en los archivos y directorios.

2.3 Propuesta de solución.

Después de realizar un estudio de las principales herramientas existentes para la verificación de la integridad de los ficheros, se concluye en la necesidad de desarrollar las funcionalidades necesarias para monitorizar la integridad de los ficheros en los servidores administrados desde HMAST reutilizando la herramienta AFICK.

El sistema permitirá gestionar las reglas por las cuales se chequearán los ficheros. Las reglas están compuestas por propiedades, las cuales se corresponden a las características de los ficheros descritas en el capítulo anterior. Además, permitirá visualizar los reportes generados como resultado de un chequeo y programar a través del cron de Unix la automatización del proceso de chequeo cada cierto tiempo.

Como funcionalidades complementarias permitirá configurar la aplicación de acuerdo con las preferencias del administrador, donde se podrá escoger la ruta donde se guarda el archivo de configuración, el archivo de base de datos y los reportes generados. Además, las configuraciones generales donde se establece la forma en que se realizan los chequeos de integridad, la profundidad de los mismos, si se excluyen algunos tipos de ficheros entre otros.

Buscando la flexibilidad en la configuración del sistema, la propuesta utiliza una arquitectura en n-capas orientada al dominio como está definida en la concepción HMAST, lo cual simplifica la comprensión y organización del sistema, reduciendo las dependencias de forma que las capas más bajas no sean conscientes de ningún detalle o interfaz de las superiores, además fomenta la reutilización. La arquitectura propuesta permitirá la interoperabilidad en entornos distribuidos con un nivel de abstracción superior, además de lograr una interfaz de usuario más flexible y permitir que la aplicación sea más simple y escalable.

2.4 Requerimientos del sistema.

Las condiciones que el sistema debe cumplir o capacidad que debe tener con el objetivo de establecer un entendimiento común entre el usuario y el proyecto de software son los requerimientos. El propósito de su gestión es establecer un entendimiento común entre el usuario y el desarrollador de software. Para el desarrollo de esta investigación se planifican dos iteraciones, siendo la mayoría de los requisitos identificados de prioridad alta y media.

Durante la captura de requisitos se confecciona la Lista de Reserva del Producto (LRP), en la cual se definen las funcionalidades que tendrá el producto en forma de requisitos técnicos y de negocio. Es una lista priorizada y garantiza la organización de los requisitos funcionales y no funcionales a partir de la prioridad que tengan para el desarrollo del sistema, de igual forma facilita el trabajo para la confección de las Historias de Usuario.

Seguidamente se muestra la LRP de la propuesta de solución de la presente investigación.

Asignado a	Ítem *	Descripción	Estimación (semanas)	Estimado por
		Prioridad **Alta**		
Yanelsi Marén-Programador	1	Instalar el módulo.	0,3	ANA
Yanelsi Marén-Programador	2	Desinstalar el módulo.	0,2	ANA
Yanelsi Marén-Programador	3	Crear alias.	0,3	ANA
Yanelsi Marén-Programador	4	Mostrar alias.	0,1	ANA
Yanelsi Marén-Programador	5	Modificar alias.	0,1	ANA
Yanelsi Marén-Programador	6	Eliminar alias.	0,2	ANA
Yanelsi Marén-Programador	7	Crear reglas de integridad.	1,2	ANA
Yanelsi Marén-Programador	8	Mostrar reglas de integridad.	0,2	ANA
Yanelsi Marén-Programador	9	Modificar reglas de integridad.	0,2	ANA
Yanelsi Marén-Programador	10	Eliminar reglas de integridad.	0,2	ANA
Yanelsi Marén-Programador	11	Crear archivo de base de datos.	0,01	ANA
Yanelsi Marén-Programador	12	Actualizar archivo de base de datos.	0,01	ANA
Yanelsi Marén-Programador	13	Chequear las reglas establecidas.	0,01	ANA
Yanelsi Marén-Programador	14	Emitir reporte sobre las inconsistencias detectadas en un chequeo.	0,2	ANA
Yanelsi Marén-Programador	15	Mostrar reporte sobre las inconsistencias detectadas en un chequeo.	0,1	ANA
		Prioridad **Media**		
Yanelsi Marén-Programador	16	Notificar vía correo el resultado de un chequeo a través del cron.	0,5	ANA
Yanelsi Marén-Programador	17	Mostrar dirección de correo	0,01	ANA
Yanelsi Marén-Programador	18	Añadir dirección de correo	0.01	ANA
Yanelsi Marén-Programador	10	Editar dirección de correo	0.01	ANA
Yanelsi Marén-Programador	20	Eliminar dirección de correo	0,01	ANA
Yanelsi Marén-	21	Mostrar cuál es el archivo de	0,1	ANA

Programador		configuraciones		
Yanelsi Marén-Programador	22	Especificar cuál es la ruta del archivo de configuraciones.	0,1	ANA
Yanelsi Marén-Programador	23	Mostrar cuál es la ruta del archivo de base de datos.	0,1	ANA
Yanelsi Marén-Programador	24	Especificar cuál es el archivo de base de datos.	0,1	ANA
Yanelsi Marén-Programador	25	Mostrar la ruta dónde guardar los reportes.	0,1	ANA
Yanelsi Marén-Programador	26	Especificar dónde guardar los reportes.	0,1	ANA
Yanelsi Marén-Programador	27	Mostrar macros.	0,1	ANA
Yanelsi Marén-Programador	28	Editar las macros.	0,1	ANA
RNF (Requisitos No Funcionales)				
Yanelsi Marén-Programador	1	Usar como framework de desarrollo Spring.	*Diseño*	
Yanelsi Marén-Programador	2	Utilizar como entorno de desarrollo integrado Netbeans.		
Yanelsi Marén-Programador	3	Utilizar como lenguaje de programación Java		
Yanelsi Marén-Programador	4	Usar como patrón arquitectónico N-capas orientado al dominio.		
Yanelsi Marén-Programador	5	Presentar simplicidad y facilidad de uso.	*Usabilidad*	
Yanelsi Marén-Programador	6	Las funcionalidades deben ser comprensibles al usuario.		
Yanelsi Marén-Programador	7	Proporcionar mantenimiento a través de versiones posteriores.	*Soporte*	
Yanelsi Marén-Programador	8	Capacidad de interacción con varios usuarios.	*Eficiencia*	

2.5 Historias de Usuario.

Las Historias de Usuario (HU) constituyen la técnica utilizada en SXP para especificar los requisitos del software, lo que equivaldría a los casos de uso en la metodología RUP [32]. Las HU 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 [33].

A continuación se exponen las Historias de Usuario de alta prioridad para el sistema.

2.5.1 HU: Instalar y desinstalar AFICK.

Historia de Usuario	
Número: HMAST_MMIAS_1	**Nombre Historia de Usuario:** Instalar y desinstalar AFICK.
Modificación de Historia de Usuario Número: 1	
Usuario: Yanelsi Marén Barrera	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0,5 semanas
Riesgo en Desarrollo: Medio	**Puntos Reales:** 0,2 semanas
Descripción: permite instalar y desinstalar la aplicación AFICK.	
Observaciones: La aplicación brinda la posibilidad al usuario de instalar AFICK es caso de que no se encuentre instalado. Para la instalación se utiliza el comando ***apt-get install afick***. El cual instala la herramienta desde el repositorio personalizado de HMAST. Antes de iniciar el proceso de instalación, la aplicación permite al usuario la posibilidad de seleccionar la ruta donde guardar los ficheros de configuración, el archivo de base de datos y los registros: ➢ **Fichero de configuración:** es un archivo con todas las configuraciones de la aplicación. Dicho archivo guarda las reglas y los parámetros generales del módulo (macros y directivas). Ruta por defecto (/etc). ➢ **Archivo de base de dados:** es un archivo que contiene el resultado de una función resumen de los archivos y directorios a los cuales se les establece las reglas de integridad. Dicha función se utiliza para verificar si los archivos o directorios han sido modificados. Ruta por defecto (/var/lib/afick/). ➢ **Registros:** guarda los registros generados en los chequeos de integridad. Ruta por defecto (/var/lib/afick/archive). Si el usuario no indica una ruta o directorio para ninguno de los casos, se toman las rutas por defecto. Además se le sugieren determinadas reglas de chequeo para que se establezca inmediatamente después de la configuración. Para desinstalar la aplicación se utiliza el comando ***apt-get remove afick***. A través del cual se desinstalará la herramienta AFICK. Una vez desinstalado el sistema pasa a estado desactivado, por este motivo, si el usuario desea interactuar con la aplicación la misma solicitará la instalación de la herramienta AFICK.	
Prototipo de interfaz: ver anexo 1.	

2.5.2 HU: Gestionar alias de reglas de integridad.

Historia de Usuario

Número: HMAST_MMIAS_2	**Nombre Historia de Usuario:** Gestionar alias de reglas de integridad.
Modificación de Historia de Usuario Número:1	
Usuario: Yanelsi Marén Barrera	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0,7 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 1 semana
Descripción: Se permite crear, mostrar, modificar y eliminar los alias que se añaden a las reglas de integridad. Un alias es un grupo de propiedades encapsuladas en un nombre establecido para ellas (ejemplo: **ETC**=p+i+n+u+g+md5).	
Observaciones: Primeramente los alias aparecerán mostrados por el nombre que los define y al lado las propiedades que lo conforman. Esta funcionalidad se realiza por defecto, pues cada vez que se acceda a la vista que permite gestionar los alias, estos se mostrarán como se describe anteriormente. Los mismos se podrán: ➢ **Crear:** Para crear un alias se define un nombre para él, con el cual se identificará el nuevo alias, no pueden existir 2 alias con el mismo nombre, y las propiedades que lo conforman. Se debe tener en cuenta que siempre debe existir al menos una propiedad asociada a este para poder crearlo. Se valida que un alias solo aparezca una vez en el listado, es decir si existe, se deben modificar las propiedades asociadas a este. ➢ **Modificar:** Consiste en adicionar o eliminar propiedades a un alias existente. Se debe tener en cuenta que siempre debe existir al menos una propiedad asociada a este. ➢ **Eliminar:** suprime los alias seleccionados.	
Prototipo de interfaz: ver anexo 2.	

2.5.3 HU: Gestionar archivo de BD.

Historia de Usuario	
Número: HMAST_MMIAS_3	**Nombre Historia de Usuario:** Gestionar archivo de base de datos.
Modificación de Historia de Usuario Número: 1	
Usuario: Yanelsi Marén Barrera	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0,02 semanas
Riesgo en Desarrollo: Alto	**Puntos Reales:** 0,02 semanas
Descripción: permite crear o inicializar y actualizar el archivo de base de datos.	

Observaciones: En un escenario general, se permitirá inicializar o crear el archivo de base de datos, así como actualizarlo. La **creación o inicialización** se lleva a cabo a través del comando **afick -i**, el cual crea una copia de los archivos y directorios especificados en el archivo de configuración en forma de base de datos la cual será usada para los chequeos de integridad posteriores. Dicho archivo se actualiza cada vez que el usuario desee. El proceso de actualización es parecido al de chequeo, el comando usado en este caso es **afick -u**. Del resultado de una actualización también se guarda un registro con el resultado del trabajo realizado. Cuando se realiza una actualización, se muestra el resultado de la misma en pantalla independientemente de que se genera un registro con el mismo.
Prototipo de interfaz: ver anexo 3.

2.5.4 HU: Gestionar reglas de integridad.

Historia de Usuario	
Número: HMAST_MMIAS_4	**Nombre Historia de Usuario:** Gestionar reglas de integridad.
Modificación de Historia de Usuario Número:1	
Usuario: Yanelsi Marén Barrera	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 1,8 semana
Riesgo en Desarrollo: Alto	**Puntos Reales:** 2 semanas
Descripción: Se puede crear, mostrar, modificar y eliminar las reglas a determinados archivos y/o directorios.	
Observaciones: las reglas aparecerán en forma de lista mostrando para identificarlas la ruta a la que se le aplica y los atributos y/o alias que las definen. En el caso que estas no lleven propiedades o alias, como son un tipo especial de reglas, solamente se muestra la ruta. Esta funcionalidad se realiza por defecto, pues cada vez que se acceda a la vista que permite gestionar las reglas, estas se mostrarán como se describe anteriormente. Las reglas pueden ser de inclusión o de exclusión. Las reglas de exclusión están constituidas por una ruta y el símbolo que define qué tipo de exclusión se va a realizar: • **!**: ignora los archivos y subdirectorios	

• **=**: ignora determinadas propiedades de los archivos y subdirectorios, se le debe decir que propiedades excluir en ellas. (Ejemplo: =/home md5). Los símbolos van delante en el caso de las reglas de exclusión. Las funcionalidades antes mencionadas permitirán: ➢ **Crear:** selecciona la ruta de un archivo o directorio y le asigna propiedades o un alias. Al crear una regla de inclusión siempre debe existir una ruta y al menos una propiedad o alias asociado a esta y para el caso de las de exclusión una ruta y uno de los 2 símbolos permitidos además de los parámetros excluyentes si es el caso. Se valida que un tipo de ruta solo aparezca una vez en el listado de reglas, es decir, si la ruta ya existe se debe modificar la regla asociada a esta y no agregar una nueva regla. ➢ **Modificar:** adiciona o elimina parámetros a una regla existente. Además permite modificar la ruta que la define. Al realizársele modificaciones a una regla se valida que haya al menos un parámetro o alias asociada a esta. ➢ **Eliminar:** suprime una o varias reglas del listado. Siempre debe haber sido seleccionada al menos una.
Prototipo de interfaz: ver anexo 4.

2.5.5 HU: Realizar chequeo de integridad.

Historia de Usuario	
Número: HMAST_MMIAS_5	**Nombre Historia de Usuario:** Realizar chequeo de integridad.
Modificación de Historia de Usuario Número: 1	
Usuario: Yanelsi Marén Barrera	**Iteración Asignada:** 1
Prioridad en Negocio: Alta	**Puntos Estimados:** 0,1 semana
Riesgo en Desarrollo: Medio	**Puntos Reales:** 0,1 semanas
Descripción: permite chequear los archivos y directorios rigiéndose por las reglas definidas	
Observaciones: se puede realizar con el comando: **afick -k,** o automáticamente a través del trabajo del cron**.** Cuando se envía la orden la aplicación realiza una comparación de los archivos y directorios incluidos en las reglas con la instantánea en la base de datos, e informa de cualquier cambio en las propiedades a chequear, así verifica si algún archivo o algún elemento en el directorio ha sido modificado. De dicho chequeo se guarda un archivo de registro con el resultado del trabajo	

realizado. Además se mostrará un resumen de dicho chequeo al usuario en pantalla, independientemente del que se ha guardado. Esta operación debe poderse ejecutar cada vez que el usuario lo desee.

Si es activado el **trabajo del cron**, deben de configurarse los elementos fundamentales para su funcionamiento. En esta sesión se establece la dirección de correo electrónico, la cantidad de líneas a enviar, y cuando se quiere recibir un correo: si cada vez que se realice un chequeo de integridad, o solamente cuando hayan cambios en el sistema de ficheros.
Para activar el trabajo del cron se requiere de determinados parámetros, los cuales se engloban en una sección definida como macros. Entre las características o parámetros que definen las macros se encuentran:

- **Lines:** Es usado para limitar longitud de los correos electrónicos enviados a través del cron.
- **Mailto:** es usado para enviar el resultado del trabajo del cron por correo electrónico.
- **Report:** es usado para habilitar y deshabilitar el envío de correo electrónico, pues si su valor es 1 se envían los correos con el resultado del trabajo del cron y si es 0 no.
- **Verbose:** se usa para definir cuándo enviar un email, si su valor es 1 se enviará un correo diario, si es 0 se enviara un correo cada vez que se encuentren cambios en los ficheros.
- **Nice:** se usa para establecer el nivel del trabajo del cron, el cual pude ser desde muy bajo si su valor es 19 hasta normal si su valor es 10 o inferior.
- **Batch:** es usado para definir si se realiza o no el trabajo del cron, si su valor es 1 es realizado y pueden realizarse las configuraciones de los parámetros anteriormente expuestos y si es cero, no se realiza el trabajo del cron.
- **Mailhost:** se utiliza para definir el servidor de correo que se utilizará para enviar los correos.
- **Mailauth:** solamente se utiliza si el servidor de correo requiere autenticación, para esto se establece un usuario y la contraseña. Si se desea enviar los correos a un servidor local esta macro se desactiva.

Prototipo de interfaz: ver anexo 5.

2.6 Plan de iteraciones.

Puntualizadas las Historias de Usuario por parte del cliente, y estimado el esfuerzo requerido para la realización de las mismas, se procede a realizar la planificación de las etapas de implementación del sistema. Este plan agrupa HU por iteraciones, definiendo en cuál de estas

serán desarrolladas en el proceso de implementación [34]. En relación con lo antes mencionado se decide implementar el sistema en dos iteraciones, las cuales se describen a continuación:

- **Iteración 1:** tiene como objetivo dar cumplimiento a las Historias de Usuario del 1 al 5. Estas han sido calificadas de prioridad alta, por lo que son de gran relevancia para la aplicación, ya que ellas constituyen el eje central de la estructura básica del sistema. Además del correcto funcionamiento de ellas dependen el resto de las funcionalidades. En esta primera etapa se realizará la instalación y configuración del módulo.
- **Iteración 2:** tiene como objetivo dar cumplimiento a las Historias de Usuario 6 y 7. Con la implementación de estas se culmina con las funcionalidades secundarias del sistema propuesto, permitiendo la monitorización de la integridad de los archivos. Una vez terminada esta iteración se tendrá la aplicación lista para su entrega.

A continuación se muestra el Plan de Iteraciones en el cual se resumen las iteraciones explicadas anteriormente:

Iteración	Descripción	Orden de las HU a implementar	Duración total
1	La implementación de las Historias de Usuario de mayor prioridad para la aplicación donde se implementarán las funcionalidades básicas.	1, 2, 3, 4, 5	3,02 semanas
2	La implementación de las Historias de Usuario que completarán las funcionalidades de la aplicación.	6,7	1, 3 semas

2.7 Estilo arquitectónico.

Los estilos arquitectónicos definen la estructura de un software, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo, con el objetivo de facilitar la tarea del diseño de tal sistema [35]. A continuación se aborda sobre el estilo arquitectural que comprende la solución propuesta.

2.7.1 Arquitectura en n-capas orientada al dominio.

La arquitectura se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades de las funcionalidades que implementan [36].

La vista de arquitectura lógica de un sistema puede considerarse como un conjunto de servicios relacionados agrupados en diversas capas. El objetivo de esta arquitectura es estructurar de una forma limpia y clara la complejidad de una aplicación empresarial basada en las diferentes capas de la arquitectura. El patrón n-capas distingue diferentes capas y sub-capas internas en una aplicación, delimitando la situación de los diferentes componentes por su tipología [37].

Adicionalmente, la separación de capas ayuda en el despliegue de un sistema distribuido, permitiendo que diferentes capas sean situadas de forma flexible en diferentes servidores o clientes, de manera que se minimice el exceso de comunicación y se mejore el rendimiento [38].

Cada capa de la aplicación contendrá una serie de componentes que implementan la funcionalidad de dicha capa.

2.8 Arquitectura del módulo.

Una vez conocida la arquitectura del sistema base descrita anteriormente (véase sección 2.1) y enunciadas las características principales de la arquitectura n-capas orientada al dominio se describe la arquitectura del módulo, la cual cuenta con cuatro capas que se describen a continuación:

Aplicación: está compuesta por las interfaces IMmiasAppService y la clase que la implementa MmiasAppService, la cual se encarga de convertir los objetos DTO que recibe por parámetro en entidades del dominio para enviarlos a la capa de dominio.

Dominio: contiene los contratos de repositorios donde se encuentra la interfaz con las funcionalidades que implementan los repositorios (IMmiasRepositorys). Además, contiene los servicios donde se realizan las validaciones suficientes para que a los repositorios llegue la información correcta y necesaria. También en esta capa se encuentran las entidades del dominio (Mmias, Alia, Regla, etc.) donde se agrupan los datos que deben persistir una vez accedida la herramienta AFICK a través del repositorio.

Repositorio: en esta capa se encuentra la clase MmiasRepository la cual implementa las funcionalidades de la interfaz IMmiasRepository (contratos de repositorios) situada en la capa de dominio. La misma es la encargada de leer y escribir en el fichero de configuración de AFICK.

2.9 Modelo de Diseño.

El Modelo de Diseño es un modelo de objetos que centra su atención en la parte física del sistema. Se enfoca particularmente en el impacto que provocan en el sistema los requerimientos funcionales y no funcionales para describir la realización física de las HU atendiendo además a otras restricciones que se puedan presentar. Se considera una base para la posterior actividad en el ciclo de vida del desarrollo de software de SXP, la implementación. Es un artefacto considerado de enorme utilidad en esta etapa gracias a que construye una arquitectura estable y sirve de abstracción al modelo de implementación [40].

En este epígrafe se recogerán los elementos más destacados dentro del modelo de diseño como es el caso de los patrones más relevantes utilizados y el diagrama de paquetes.

2.9.1 Patrones del diseño.

Los patrones de diseño resuelven problemas comunes a partir de soluciones probadas mediante un diseño orientado a objeto facilitando la reutilización de clases ya existentes. Dentro de los elementos más importantes que componen un patrón, se destacan el nombre del patrón, el problema que resuelve y la solución al mismo. El framework Spring implementa una serie de patrones de diseño que hacen que su arquitectura sea suficientemente robusta y a la vez flexible como para adaptarse a los casos más complejos. A continuación se especifican los patrones presentes dentro de la solución propuesta:

Creador: “El patrón Creador aporta un principio general para la creación de objetos, una de las actividades más frecuentes en programación” [41]. Como su nombre lo indica es el que crea y guía la asignación de responsabilidades relacionadas con la creación de objetos, encontrando un creador que debemos conectar con el objeto producido en cualquier evento. Este patrón se utilizó en la Capa de Aplicación cuando para la conversión de clases de tipo DTO a entidad, se crean instancias de las entidades, debido a que las clases DTO contienen los datos de inicialización de estas.

Experto: “El patrón Experto posibilita una adecuada asignación de responsabilidades facilitando la comprensión del sistema, su mantenimiento y adaptación a los cambios con reutilización de componentes” [42]. Una clase contiene toda la información necesaria para realizar la labor que tiene encomendada, lo cual es un principio básico que suele ser útil en el diseño orientado a objetos. El uso de este patrón brinda beneficios como la conservación del encapsulamiento y el soporte de un bajo acoplamiento y una alta cohesión. En el sistema se hace uso de este patrón en casos como la verificación en el paquete de servicio, perteneciente a la Capa de Dominio, de la existencia de las entidades, donde esta responsabilidad es asignada a la Capa de Persistencia.

Bajo acoplamiento: “El patrón Bajo acoplamiento es una medida de la fuerza con que una clase se relaciona con otras, porque las conoce y recurre a ellas; una clase con bajo acoplamiento no depende de muchas otras, mientras que otra con alto acoplamiento presenta varios inconvenientes: es difícil entender cuando está aislada, es ardua de reutilizar porque requiere la presencia de otras clases con las que esté conectada y es cambiante a nivel local cuando se modifican las clases afines” [43]. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. El uso de este patrón permite que las clases no se afecten por cambios de otros componentes, haciendo posible que sean fáciles de entender y de reutilizar. También las inyecciones de dependencia proporcionadas por el framework Spring, permiten el uso de este patrón en el sistema.

Alta cohesión: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. Se hizo necesario la utilización de este patrón en el software en cuestión con el fin de controlar la complejidad de cada clase utilizada para mantener un buen comportamiento de las mismas, por esta razón, las que se identificaron con una amplia cantidad de funcionalidades se dividieron en otras clases siguiendo el propósito de distribuir de forma equitativa el peso de la complejidad, manteniendo además la coherencia entre ellas.

Instancia única (Singleton): “Garantiza que una clase solo tenga una única instancia, y proporciona un punto de acceso global a ella” [44]. Este patrón establece la creación de una

instancia única de un objeto, para ser accedida luego por un método invocado, además es muy flexible al realizar modificaciones para controlar el número de las mismas.

Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El control realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos, se puede evidenciar en la conexión a un servidor ya que con una única instancia de este objeto se realizan todas las operaciones sobre el servidor.

2.9.2 Diagramas de paquetes.

El diagrama de paquetes representa la interacción de los subpaquetes y las clases del módulo, este contiene las relaciones entre ellas así como los atributos y métodos que componen cada una [45]. En el sistema que se está proponiendo se presenta a continuación el diagrama de paquetes que conforma la solución propuesta para la historia de usuario instalar y desinstalar AFICK que se encuentra en el módulo de integridad de ficheros.

A continuación se describe el diagrama con el objetivo de comprender como se realiza la interacción entre las capas.

En la capa de aplicación (*application*) se encuentra la clase interfaz IConfigurationAppService que contiene los métodos que permiten el acceso desde la capa de presentación, los cuales son implementados por la clase ConfigurationAppService que es responsable de adaptar la información que le llega desde la interfaz de usuario a los requerimientos de los servicios del dominio (*domain*), estos servicios son accedidos a través del atributo **mmiasService** que contiene esta clase, en esta capa se encuentra el paquete **DTO** que contiene clases que representan la información que llega desde la capa de presentación.

En la capa de dominio (*domain*) existen tres subpaquetes ***Entitys***, ***Service*** y ***RepositorysContracts***. En el subpaquete *Entitys* se encuentra la clase Mmias que representa todo el sistema en el dominio, por tanto almacena toda la información una vez que se instala AFICK y se cargan los datos. En el subpaquete *Service* se encuentra la clase interfaz IConfigurationService que contiene métodos que son accedidos desde la capa de aplicación los cuales son implementados por la clase ConfigurationService que es la responsable de realizar las

validaciones de los datos antes de realizar la escritura o lectura en/del repositorio, el acceso al repositorio se realiza a través del atributo mmiasRepository encontrado en esta clase. En el subpaquete **repositorysContracts** se encuentra la clase interfaz IMmiasRepository que tiene definidos los métodos que son accedidos desde el subpaquete Service los cuales son implementados por la clase MmiasRepository encontrada en el paquete ***Persistens***, esta clase es responsable de cargar y salvar la información en los repositorios.

Los diagramas de paquetes que representan el resto de las Historias de Usuarios presentan el mismo flujo de comunicación entre clases que se describió anteriormente independientemente de las funcionalidades que estas contengan. A continuación se representa el diagrama de paquetes descrito anteriormente.

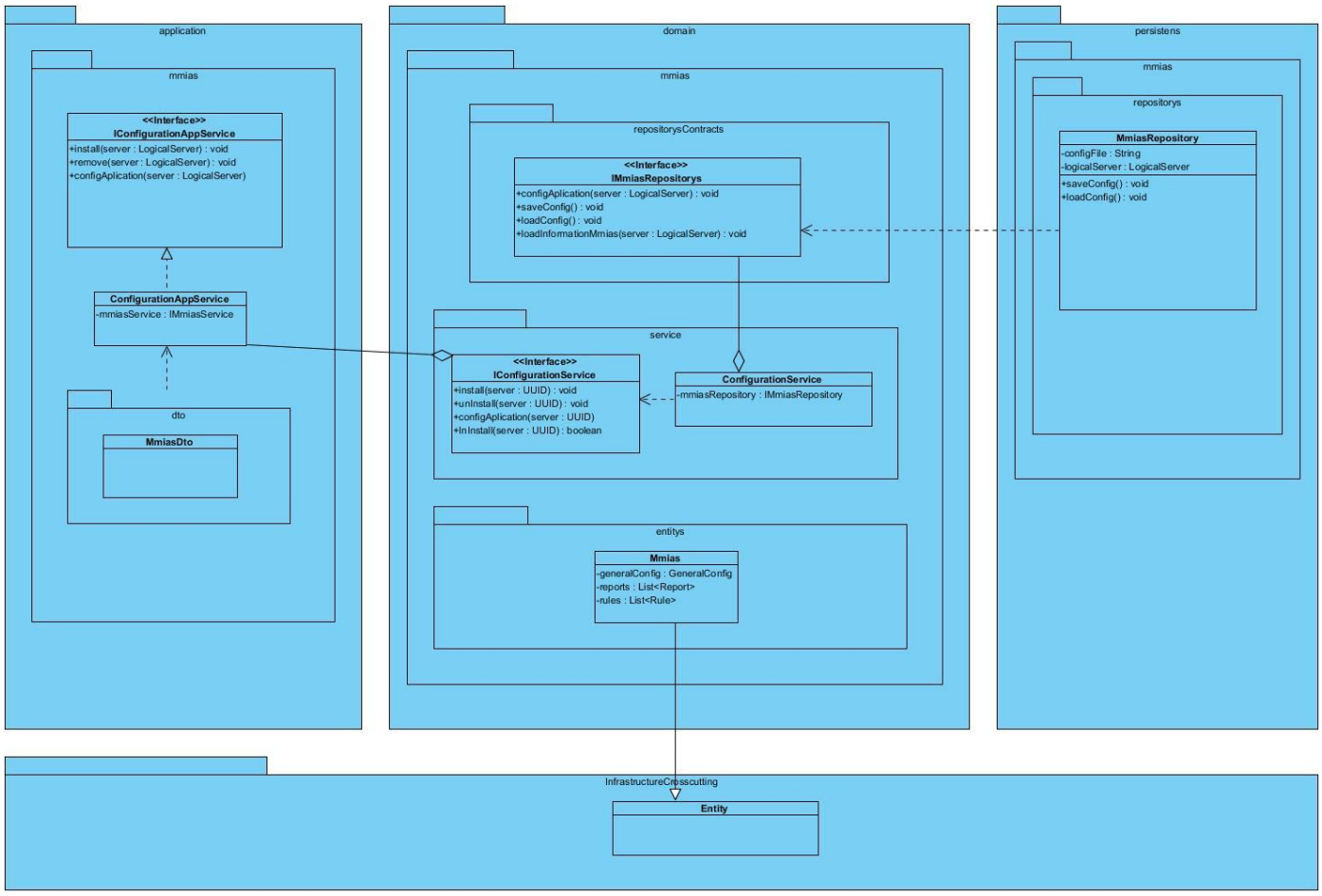


Figura 3: Diagrama de paquetes

Conclusiones parciales.

Hasta el momento, la investigación realizada ha llegado al punto donde la utilización de SXP ha guiado satisfactoriamente el desarrollo del software, encontrándose este listo para la implementación. Se siguió un conjunto de pasos para obtener una concepción del sistema que cumpla con los objetivos y expectativas marcados, de forma tal que se incluyen en esta sección los artefactos generados en correspondencia con los flujos de trabajo desarrollados.

Capítulo 3: Implementación y pruebas del sistema.

En el presente capítulo se refleja el flujo de actividades que conforman la implementación y pruebas de la solución propuesta. Describiéndose los estándares de código, las Tareas de Ingeniería, y las pruebas realizadas.

3.1 Estándares de código.

Un estándar de codificación comprende todos los aspectos de la generación de código. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Los estándares de codificación permiten una mejor integración entre las líneas de producción y establece pautas que conllevan a lograr un código más legible y reutilizable, de tal forma que se pueda aumentar su capacidad de mantenimiento a lo largo del tiempo [46].

Nomenclatura de las clases.

Los nombres de las clases comienzan con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación **PascalCasing**. Con solo leerlo se reconoce el propósito de la misma, pues cada palabra comienza con letra mayúscula [47].

Ejemplo: **RuleExcludeProperty**

Nomenclatura de las funciones.

El nombre a emplear para las funciones se escribe con la primera palabra en minúscula, en caso de que sea un nombre compuesto se empleará notación **CamelCasing**, y con solo leerlo se reconoce el propósito de la misma.

Ejemplo: **addRule**

Estándar para nombrar las variables.

El nombre de las variables se escribe con primera palabra en minúscula, si es un nombre compuesto se utilizará notación **CamelCasing**.

Ejemplo: **mmiasService**

Nomenclatura de los componentes:

Todos los paquetes comienzan con cu.uci.hmast.xxx.yyy.zzz.kkk

xxx → Capa donde se encuentra (presentación, aplicación, dominio, persistencia).

yyy →Nombre del módulo (postfix, dhcp3, mmias, etc).

zzz →Elementos que pueden contener los componentes verticales (Entitys, repositorys, etc).

kkk → Clases o subpaquetes.

Ejemplo: **cu.uci.hmast.domain.mmias.Entity.Report**

Nomenclatura de los comentarios.

Los comentarios deben ser claros y precisos de forma tal que se entienda el propósito de lo que se está desarrollando. En caso de ser una función complicada se comentaría dentro de la misma para lograr una mejor comprensión del código.

Ejemplo:

```
    /**
     * Permite adicionar una regla al servidor que se corresponda con el identificador que se pasa por parámetro
     * @param identificador de un servidor
     * @param una regla
     * @throws JSchException
     * @throws IOException
     * @throws EDuplicateObject si la regla ya existe
     * @throws ELogicalServerNotExist
     * @throws EInvalidRule si lo que se recibe por parámetro no es una regla
     * @throws EPaperInvalid si el archivo es incorrecto.
     * @throws EInvalidObject si el servidor que se recibe por parámetro no existe
     * @throws EInvalidPath si la ruta no es valida
     * @throws EInvalidSymbol si el símbolo recibido no se corresponde con los permitidos en las reglas
     * @throws EExistingItem
     */
    void addRule(UUID server, RuleDto rule) throws JSchException, IOException, InterruptedException, EDuplicateObject, ELogicalServerNotExist, EInvalidRule, EPaperInvalid, EInvalidObject, EInvalidPath, EInvalidSymbol, EExistingItem;
```

3.2 Tareas de Ingeniería.

Las Tareas de Ingeniería agrupan las actividades a realizar por Historias de Usuario. Las mismas complementan o facilitan el desarrollo de cada una [48]. A continuación se muestran las tareas planificadas para las Historias de Usuario de la solución propuesta.

No	Nombre de la tarea	HU	Tipo tarea
1	Estudio de los comandos y opciones para instalar la aplicación.	1	Desarrollo
2	Estudio de los comandos y parámetros para visualizar reportes.	6	Desarrollo
3	Estudio de las directivas y las macros.	7	Desarrollo

Tabla 2: Tareas de ingeniería

La realización de las Tareas de Ingeniería facilitó la implementación de las HU vinculadas a estas, además de contribuir a elevar el nivel de conocimientos sobre la herramienta.

3.3 Pruebas de software.

El instrumento adecuado para determinar el estado de la calidad de un producto de software es el proceso de pruebas. En el mismo se ejecutan pruebas dirigidas a componentes del sistema de software en su totalidad, con el objetivo de medir el grado en que la aplicación cumple con los requerimientos [49].

Este no es un proceso que se realiza una vez desarrollado el software, sino que debe efectuarse en cada una de las etapas de desarrollo. La creciente inclusión del software como un elemento más de muchos sistemas y la importancia de los costos asociados a un fallo del mismo han motivado la creación de pruebas más minuciosas y bien planificadas.

Descripción de las pruebas.

Con diferentes objetivos, en variados escenarios o niveles de trabajos se enfocan las pruebas en las cuales se distinguen los diferentes niveles que a continuación se mencionan:

- Prueba de desarrollador

- Prueba independiente
- Prueba de unidad
- Prueba de integración
- Prueba de sistema
- Prueba de aceptación

La **Prueba de Unidad** se enfoca a los elementos comprobables más pequeños del software, estos pueden ser clases, métodos, propiedades, componentes, etc. La prueba de unidad siempre está orientada a Caja Blanca es preciso probar el flujo de datos de la interfaz antes de iniciar cualquier otra prueba. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. El diseño de casos de prueba de una unidad comienza una vez que se ha desarrollado, revisado y verificado en su sintaxis el código a nivel fuente.

Una de las técnicas para ejecutar las pruebas de caja blanca es la del “Camino básico”, el resultado de esta técnica es una medición cuantitativa de la complejidad de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

3.3.1 Aplicación de pruebas de unidad.

Para realizar la prueba es necesario primeramente el cálculo de los valores de la complejidad ciclomática de los procedimientos a los cuales se le va a aplicar la prueba. A continuación se muestran cada uno de los pasos realizados para la funcionalidad adicionar una regla.

En esta se tiene en cuenta el código correspondiente a la funcionalidad en la capa de aplicación, se divide el código en bloques, como se muestra en la Figura 4 y posteriormente, en base a la numeración obtenida, se representa el flujo de control lógico mediante la notación del grafo de flujo, que se aprecia en la Figura 5.

```
public void addRule(UUID server, RuleDto rule) {
1   if (rule instanceof RuleIncludeDto) {
        RuleIncludeDto includeDto = (RuleIncludeDto) rule;
        List<String> alia = includeDto.getAliaDto();;

        List<Property> properties = new LinkedList<Property>();
2       for (int i = 1; i < includeDto.getPropertiesDto().size(); i++) {
            Property p = Property.valueOf(includeDto.getPropertiesDto().get(i).name());
            properties.add(p);
        }
        RuleInclude include = new RuleInclude(includeDto.getId(), alia, properties, includeDto.getRute());
        mmiasService.addRule(server, include);
        break;
    }
3   if (rule instanceof RuleExcludeFileDto) {
        RuleExcludeFileDto excludeFDto = (RuleExcludeFileDto) rule;
4       RuleExcludeFile excludeF = new RuleExcludeFile(excludeFDto.getSymbol(), excludeFDto.getRute());
        mmiasService.addRule(server, excludeF);
        break;
    }
5   if (rule instanceof RuleExcludePropertyDto) {
        RuleExcludePropertyDto excluedePDto = (RuleExcludePropertyDto) rule;
6       RuleExcludeProperty excluedeP = new RuleExcludeProperty(excluedePDto.getSymbol(), excluedePDto.getAlias(),
        excluedePDto.getPropertys(), excluedePDto.getRute());
        mmiasService.addRule(server, excluedeP);
    }

7}
```

Figura 4: Fragmento de código al que se aplica la prueba

Seguidamente se representa el grafo del código anterior, el cual está compuesto por los componentes siguientes:

- **Nodo:** son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión (Nodos predicados). Los nodos que no están asociados se utilizan al inicio y final del grafo.

- **Aristas:** son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento.

- **Regiones:** son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo como una región más. Las regiones se enumeran siendo la cantidad de re-

giones equivalentes a la cantidad de caminos independientes del conjunto básico de un procedimiento.

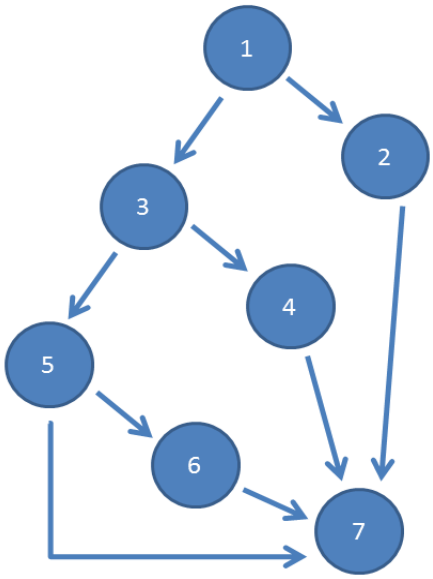


Figura 5: Grafo asociado al código de prueba

Luego de haber construido el grafo se realiza el cálculo de la complejidad ciclomática, mediante las tres fórmulas de análisis de complejidad del algoritmo, las cuales tienen que arrojar el mismo resultado para asegurar que el cálculo de la complejidad es correcto. Fórmulas para calcular complejidad ciclomática:

1 – V (G) = (Aristas – Nodos) + 2.

2 – V (G) = Nodos Predicados + 1.

3 – V (G) = Regiones.

Aplicando estas fórmulas al grafo anteriormente mostrado se obtienen los siguientes resultados:

- **Calculando mediante la fórmula 1**

V(G)=(9-7)+2

V(G)=4

- **Calculando mediante la fórmula 2**

V(G)=3+1

V(G)=3

- **Calculando mediante la fórmula 3**

V(G)= 4

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede decir que la complejidad ciclomática del código es 4, lo cual da un límite superior para el número de caminos independientes que componen el conjunto básico y, consecuentemente, un valor límite superior para el número de pruebas que se deben diseñar y ejecutar para garantizar que se cubren todas las sentencias de los procedimientos. Es necesario representar los caminos básicos por los que puede recorrer el flujo.

Camino básico # 1: 1-2-7

Camino básico # 2: 1-3-4-7

Camino básico # 3: 1-3-5-6-7

Camino básico # 4: 1-3-5-7

Después de haber extraído los caminos básicos, se procede a ejecutar los casos de pruebas, se debe realizar al menos un caso de prueba por cada camino básico.

Caso de prueba para el camino básico # 1
Descripción: Se adiciona una regla.
Entrada: Se pasa como parámetro el identificador del servidor (server) y una regla (RuleIncludeDto) que tiene los siguientes atributos: Rute: /home/maren/Escritorio List<Values>:p+n+g+u+md5 server: 344-ewr-4324-4f2e2-24f824cv-42524
Resultado esperado: El servidor lógico ya existe en el mapa, al existir atributos correspondientes a las reglas en el mapa temporal estos se integran a los nuevos parámetros, todos los atributos son válidos por lo que se realiza la adición de la regla.
Resultado obtenido: El camino se ha recorrido satisfactoriamente.

Caso de prueba para el camino básico # 2
Descripción: Se adiciona una regla.
Entrada: Se pasa como parámetro el identificador del servidor (server) y una regla (RuleExcludeFileDto) que tiene los siguientes atributos: Rute: /home/maren/Escritorio/prueba Symbol: ! server: 344-ewr-4324-4f2e2-24f824cv-42524
Resultado esperado: El servidor lógico ya existe en el mapa, al existir atributos correspondientes a las reglas en el mapa temporal estos se integran a los nuevos parámetros, todos los atributos son válidos por lo que se realiza la adición de la regla.
Resultado obtenido: El camino se ha recorrido satisfactoriamente.

Caso de prueba para el camino básico # 3
Descripción: Se adiciona una regla.
Entrada: Se pasa como parámetro el identificador del servidor (server) y una regla (RuleExcludePropertyDto) que tiene los siguientes atributos: Rute: /home/maren/Escritorio/ Symbol: = List<Values>:p server: 344-ewr-4324-4f2e2-24f824cv-42524
Resultado esperado: El servidor lógico ya existe en el mapa, al existir atributos correspondientes a las reglas en el mapa temporal estos se integran a los nuevos parámetros, todos los atributos son válidos por lo que se realiza la adición de la regla.
Resultado obtenido: El camino se ha recorrido satisfactoriamente.

Caso de prueba para el camino básico # 4

Descripción: Se adiciona una regla.
Entrada: Se pasa como parámetro el identificador del servidor (server) y una regla que tiene los siguientes atributos: Rule=Null server: 344-ewr-4324-4f2e2-24f824cv-42524
Resultado esperado: El servidor lógico ya existe en el mapa, al existir atributos correspondientes a las reglas en el mapa temporal estos no se integran a los nuevos parámetros debido a que los atributos no son válidos por lo que no se realiza la adición de la regla.
Resultado obtenido: El camino se ha recorrido satisfactoriamente. Se lanza la excepción EInvalidRule.

3.3.2 Pruebas por parte del cliente.

La norma cubana NC-ISO/IEC 9126-1:2005 elaborada por la Oficina Nacional de Normalización, plantea que la calidad del producto del software se debe evaluar usando un modelo de calidad definido. El modelo de calidad para la calidad interna y externa categoriza los atributos en seis características: la funcionalidad, la confiabilidad, la usabilidad, la eficiencia, la mantenibilidad y la portabilidad. Esta última abarca atributos como son la adaptabilidad, inestabilidad, coexistencia, reemplazabilidad, portabilidad y conformidad. Específicamente en el módulo desarrollado se verificará la adaptabilidad. Según la norma citada, la adaptabilidad se define como la capacidad del producto de software de ser adaptado a los ambientes especificados sin aplicar acciones o medios de otra manera que aquellos suministrados con el propósito de que el software cumpla sus fines [50].

Con el fin de demostrar el cumplimiento del objetivo planteado y la satisfacción del cliente con el resultado obtenido, se realizaron pruebas para valorar la calidad del módulo implementado. Se probaron cada una de las funcionalidades mediante un *Main* de pruebas realizándose llamadas a estas, dentro de las cuales se pueden citar:

- **Adicionar Reglas:** pasando como parámetros el identificador del servidor al que se adicionará y la regla que se desea incluir en el mismo.

- **Adicionar Alias:** pasando como parámetros el identificador del servidor al que se adicionará y el alias que se desea incluir.

- **Realizar chequeo:** pasando como parámetros el identificador del servidor al que se le realizará dicho chequeo.

3.3.3 Resultados de las pruebas.

Las pruebas realizadas permitieron evaluar la calidad de MMIAS en las siguientes condiciones tecnológicas:

- Computadora de escritorio con GNU/Linux Nova Escritorio, procesador 1.40GHz, 1GB de RAM, 512 MB de memoria SWAP, tarjeta Ethernet a 100Mbs.

- La prueba se ejecutó sobre GNU/Linux Nova Escritorio ejecutándose sobre una máquina virtual, con 512 MB de RAM (Se seleccionó esta medida por ser común en varios OACE).

Buscar los elementos más relevantes de las pruebas es el mayor peso a la hora de analizar los resultados una vez ejecutadas. Un factor importante a tener en cuenta es que a medida que se fue desarrollado el software se fueron aplicando pruebas a nivel del programador, las cuales consisten en ir probando el código con el propósito de verificar que lo que se esté implementando funcione de forma correcta. Una vez concluida la fase de implementación, se le aplicó una iteración completa de pruebas a la aplicación, donde quedaron visitados todos los caminos independientes identificados, y fueron encontradas un total de dos no conformidades, asociadas a errores de validación.

A partir de este resultado fue realizada una segunda iteración de pruebas, con el propósito de erradicar los errores existentes y no fue encontrada ninguna no conformidad, por tanto el software se encuentra listo para ser usado. Luego de aplicar los distintos casos de pruebas, se pudo comprobar que el flujo de trabajo de cada procedimiento está correcto ya que cumplen con las condiciones necesarias que se habían planteado.

Conclusiones parciales.

Al finalizar el desarrollo de este capítulo se obtuvo un módulo completamente funcional. En este punto se encuentran implementados los elementos del diseño satisfactoriamente con los

requerimientos definidos. Se logró identificar las principales fortalezas a partir del análisis de los resultados arrojados una vez realizadas las pruebas al software.

Conclusiones.

Una vez cumplidos los objetivos se arribó a las siguientes conclusiones:

- El estudio realizado sobre las herramientas de código abierto para la monitorización de la integridad de los archivos permitió determinar que AFICK es la más adecuada para realizar esta acción en HMAST.
- El módulo desarrollado facilita la administración y configuración de la herramienta AFICK.
- La ejecución de pruebas realizadas al culminar cada nueva funcionalidad y cada iteración garantizó el desarrollo de un módulo con la calidad requerida por el cliente.

Recomendaciones.

Como resultado del desarrollo de la presente investigación se obtuvo el módulo para monitorizar la integridad de los ficheros. El cual cuenta con las funcionalidades básicas que realiza la herramienta de verificación de integridad AFICK, por lo que se recomienda para mejorar el funcionamiento del mismo los aspectos siguientes:

- Gestión de directivas (ejemplos: report_url, warn_missing_file) y macros (ejemplos: mount, nagios_config) que no se incluyen en esta versión del módulo, pero que están presentes en el fichero de configuración de AFICK por lo que pueden ser configuradas para ampliar el alcance de los chequeos de integridad.
- Desarrollar una interfaz visual que permita configurar y ejecutar la herramienta AFICK utilizando las funcionalidades implementadas.

Referencias Bibliográficas.

[1]. Zuccardi, Giovanni y David Gutiérrez, Juan. Seguridad Informática. Octubre 2006. [Citado 23 octubre 2012]. Disponible en: <http://pegasus.javeriana.edu.co/~edigital/Docs/Seguridad%20Informatica/Seguridad%20Informatica%20v1.0.pdf>

[2]. ¿Seguridad en Unix? [Citado 26 de octubre 2012]. Disponible en: <http://www2.tiendalinux.com/docs/manuales/unixsec/unixsec-1.2/node5.html>

[3]. GNU. ¿Qué es software libre? [Citado: 23 octubre 2012]. Disponible en: <http://www.gnu.org/philosophy/free-sw.es.html.>

[4]. Seguridad Informática vs Seguridad de la Información. [Citado 10 noviembre 2012]. Disponible en: <http://www.audea.com/seguridad-de-la-informacion-vs-seguridad-informatica/>

[5]. Seguridad de la Información. [Citado 5 diciembre 2012]. Disponible en: <http://www.inteco.es/blog/Seguridad/Observatorio/BlogSeguridad/>

[6]. Normas y estándares. [Citado 15 octubre 2012]. Disponible en: <http://ccia.ei.uvigo.es/docencia/SSI/normas-leyes.pdf >

[7]. Gestión de Riesgo de la seguridad Informática. [Citado 15 octubre 2012]. Disponible en: <http://protejete.wordpress.com/glosario/>

[8]. Seguridad de la Información. [Citado 5 diciembre 2012]. Disponible en: <http://www.inteco.es/blog/Seguridad/Observatorio/BlogSeguridad/.> Página 2.

[9]. López Manrique, Yuri Vladimir. Computación Forense: Una Forma de obtener para combatir y prevenir delitos informáticos. 2007.

[10]. Tripwire. [Citado 15 octubre 2012]. Disponible en: <http://www.linux-cd.com.ar/manuales/rh9.0/rhl-rg-es-9/ch-tripwire.html>

[11]. Tripwire. [Citado 15 octubre 2012]. Disponible en: <http://www.linux-cd.com.ar/manuales/rh9.0/rhl-rg-es-9/ch-tripwire.html.> Sección 19.4

[12]. iWatch. [Citado 15 octubre 2012]. Disponible en: <http://iwatch.sourceforge.net/index.html>

[13]. Frommel, Oliver. Monitorizar directorios con iWatch. Número 29.

[14]. Integrit. [Citado 10 enero 2013]. Disponible en: <http://packages.debian.org/es/sid/integrit>

[15]. AFICK. [Citado 15 octubre 2012]. Disponible en: <http://afick.sourceforge.net/index.html>

[16]. Warshawsky, Brian Linux server: los mejores trucos. Página 364.

[17]. Administración y seguridad de servidores: sistemas de archivos. [Citado 22 marzo 2013]. Disponible en: <http://www.ibt.unam.mx/jmanuel/cursoservidores/sistemasdearchivos.html>

[18]. Metodología de Desarrollo de Software. [Citado 15 enero 2013]. Disponible en: <http://www.um.es/docencia/barzana/IAGP/Iagp2.html>

[19]. G. Figueroa, Roberth, J. Solis, Camilo y Cabrera, Armando. Metodologías tradicionales vs. Metodologías Ágiles. 2011.

[20]. Peñalver Romero, Gladys Marsi, García de la Puente, Sergio y Meneses Abad, Jesús Abel. SXP, metodología de desarrollo de software. 2010.

[21]. Visual Paradigm for UML (ME). [Citado 27 abril 2012]. Disponible en: <http://www.freedownloadmanager.org/es/downloads/%20Paradigma_Visual_para_UML_%28M%C3%8D%29%20_14720_p/>

[22]. G. B, Ivar Jacobson. El proceso unificado de desarrollo de software. Habana: Félix Varela. 2004.

[23]. Welcome to NetBeans. [Citado 15 diciembre 2012]. Disponible en: <http://netbeans.org/>

[24]. Introducción a Spring Framework Java | Picando Código. [Citado 23 noviembre 2012]. Disponible en: <http://picandocodigo.net/2010/introduccion-a-spring-framework-java/>

[25]. Polanco Velasco, Marcos A. Externalización de servicios seguros. [Citado 28 noviembre 2011]. Disponible en: <http://enise.inteco.es/images/stories/Ponencias/T25/marcos%20polanco.pdf>

[26]. Melodysof. Los Lenguajes de Programación. [Citado 1 diciembre 2012]. Disponible en: <http://boards5.melodysoft.com/2007AISC0107/los-lenguajes-de-programacion-15.html>

[27]. Segura Salazar, Juan. Características de JAVA. [Citado 17 febrero 2013]. Disponible en: <http://tikal.cifn.unam.mx/~jsegura/LCGII/java3.htm>

[28]. Historia de Java. [Citado 27 enero 2013]. Disponible en: <http://www.cad.com.mx/historia_del_lenguaje_java.htm>

[29]. XML at the Apache foundation. [Citado 10 febrero 2013]. Disponible en: <http://xml.apache.org/>

[30]. Análisis y diseño de Sistemas. [Citado 22 marzo 2013]. Disponible en: <http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/P_proceso/ANALISIS_Y_DISEnO_DE_SISTEMAS/IngenieriaDeSoftware/CIS/UNIDAD%20VI/6.5.htm>

[31]. Modelo de dominio. Tecnología y Synergix.

[32]. Peñalver Romero, Gladys Marsi, García de la Puente, Sergio y Meneses Abad, Jesús Abel. SXP, metodología de desarrollo de software. 2010. Página 57.

[33]. Peñalver Romero, Gladys Marsi, García de la Puente, Sergio y Meneses Abad, Jesús Abel. SXP, metodología de desarrollo de software. 2010. Página 58.

[34]. Peñalver Romero, Gladys Marsi, García de la Puente, Sergio y Meneses Abad, Jesús Abel. SXP, metodología de desarrollo de software. 2010. Página 56.

[35]. Estilos Arquitectónicos. [Citado 15 marzo 2013]. Disponible en: <http://es.scribd.com/doc/23161581/Estilos-Arquitectonico>

[36]. Zorrilla Castro, César, Unai de la Torre Llorente, Ramos Barroso, Miguel Angle, Calvarro Nelso, Javier. Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0. 2010. 422 p. ISBN 978-84-936696-3-8. Página 22.

[37]. Zorrilla Castro, César, Unai de la Torre Llorente, Ramos Barroso, Miguel Angle, Calvarro Nelso, Javier. Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0. 2010. 422 p. ISBN 978-84-936696-3-8. Página 24.

[38]. Zorrilla Castro, César, Unai de la Torre Llorente, Ramos Barroso, Miguel Angle, Calvarro Nelso, Javier. Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0. 2010. 422 p. ISBN 978-84-936696-3-8. Página 31.

[39]. Zorrilla Castro, César, Unai de la Torre Llorente, Ramos Barroso, Miguel Angle, Calvarro Nelso, Javier. Guía de Arquitectura N-Capas orientada al Dominio con .NET 4.0. 2010. 422 p. ISBN 978-84-936696-3-8. Página 33.

[40]. Rojas, N. y Hecheverría, Y. Sistema para la gestión de los eventos científicos en la universidad de las ciencias informáticas. 2008-2009.

[41]. R. Botero Tabares. Patrones grasp y anti-patrones: Un enfoque orientado a objetos desde la lógica de programación. Entre ciencia e ingeniería, (8):161–173, 2011.

[42]. R. Botero Tabares. Patrones grasp y anti-patrones: Un enfoque orientado a objetos desde la lógica de programación. Entre ciencia e ingeniería, (8):161–173, 2011. Página 20.

[43]. R. Botero Tabares. Patrones grasp y anti-patrones: Un enfoque orientado a objetos desde la lógica de programación. Entre ciencia e ingeniería, (8):161–173, 2011. Página 22.

[44]. Patrones de diseño, análisis y diseño. Ingeniería del software. [Citado 25 marzo 2013]. Disponible en: <http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php>

[45]. Diagrama de paquetes. [Citado 20 marzo 2013]. Disponible en: <http://jms32.eresmas.net/tacticos/UML/UML07/UML0701.html>

[46]. Estándares de codificación. [Citado 27 marzo 2013]. Disponible en: <http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx>

[47]. Informatízate: Dime como programas y te diré quién eres. [Citado 5 abril 2013]. Disponible en:

<http://www.informatizate.net/articulos/dime_como_programas_y_te_dire_quien_eres_23082004.html>

[48]. Peñalver Romero, Gladys Marsi, García de la Puente, Sergio y Meneses Abad, Jesús Abel. SXP, metodología de desarrollo de software. 2010. Página 63.

[49]. R.S. PRESSMAN. Ingeniería del software un enfoque práctico. Quinta edición. Madrid. Macgraw-hill.2002.

[50]. Norma cubana ISO/IEC 9126. [Citado 20 mayo 2013]. Disponible en: <http://www.cgdc.cubaindustria.cu/products/revista-normalizacion-numero-2-3-2007.htm>

Glosario de términos.

Advenimiento: es un término vinculado al verbo advenir del latín advenïre, que se refiere a la acción de llegar, suceder o venir.

Alias: nombre que reciben un conjunto de propiedades de AFICK para ser identificadas como un todo.

Directivas: son variables almacenadas en el archivo de configuración para controlar el funcionamiento de AFICK en tiempo de ejecución según sus valores.

Directorio: carpeta o gaveta que contiene varios archivos y/o directorios. Se utilizan para organizar la información almacenada en el sistema operativo.

Dynamic Host Configuration Protocol (DHCP): Protocolo de Configuración de Hosts Dinámicos. Es un protocolo utilizado para asignar direcciones de IP dinámicas en una red.

Fichero: archivo o documento que se encuentra en un sistema de cómputo, puede ser de distintos tipos (ejemplo de texto, configuración, imagen, etc.) según el fin con el que fue creado.

Framework: es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Historias de Usuario (HU): secuencias de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de las secuencias.

Instalar: incorporar a la computadora un programa o dispositivo para ser utilizado.

Integridad: calidad de algo que no carece de ninguna de sus partes.

Kernel: núcleo. Parte esencial de un sistema operativo que provee los servicios más básicos del sistema. Se encarga de gestionar los recursos como el acceso seguro al hardware de la computadora, determina qué programa accederá a un determinado hardware, si dos o más programas quieren usarlo al mismo tiempo, entre otras.

Logs: registro oficial de eventos durante un rango de tiempo en particular.

RAM: se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo.

Ruta: enlaces para encontrar la ubicación de un fichero o directorio en el sistema operativo.

Servidor: computadora central de un sistema de red que provee servicios y recursos (programas, comunicaciones, archivos, etc.) a otras computadoras (clientes) conectadas a ella.

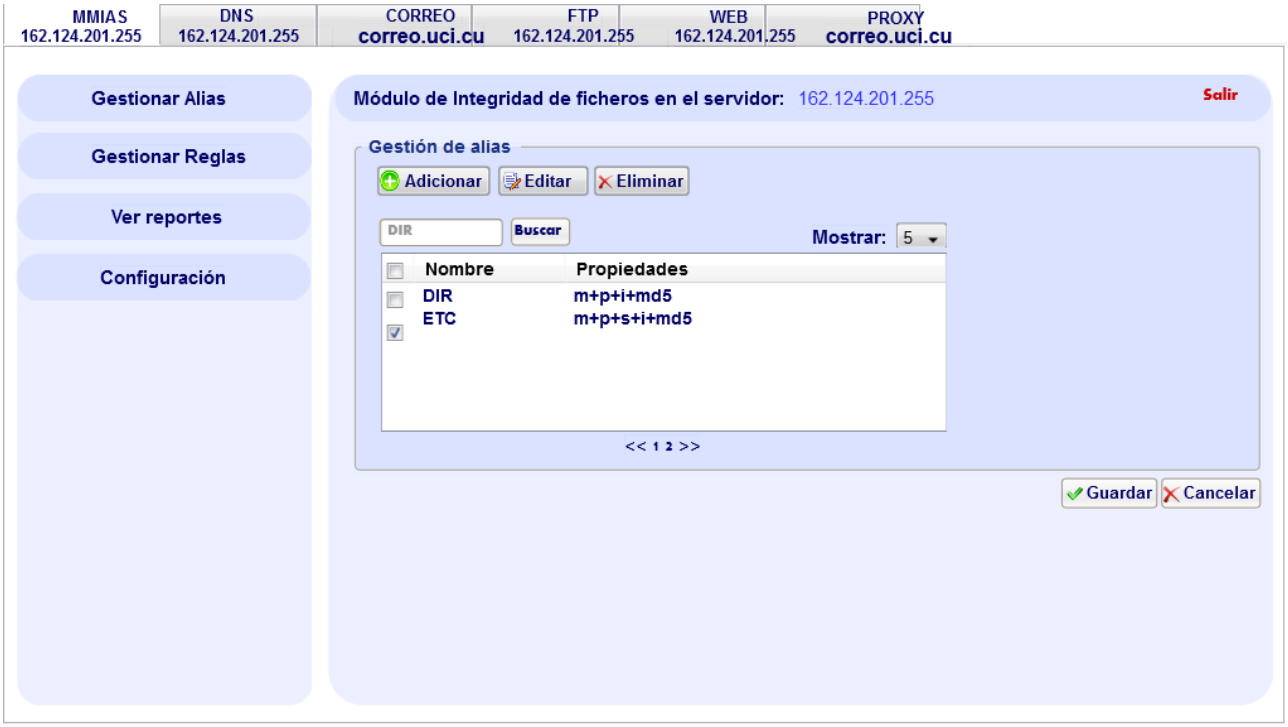
SSH: es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos, y también puede redirigir el tráfico de X para poder ejecutar programas gráficos si se tiene un servidor X (en sistemas Unix y Windows) corriendo.

Taxonomía: es la ciencia en la que se clasifican los organismos y se establecen los parámetros de diferencia, creando familias, ramas y conjuntos.

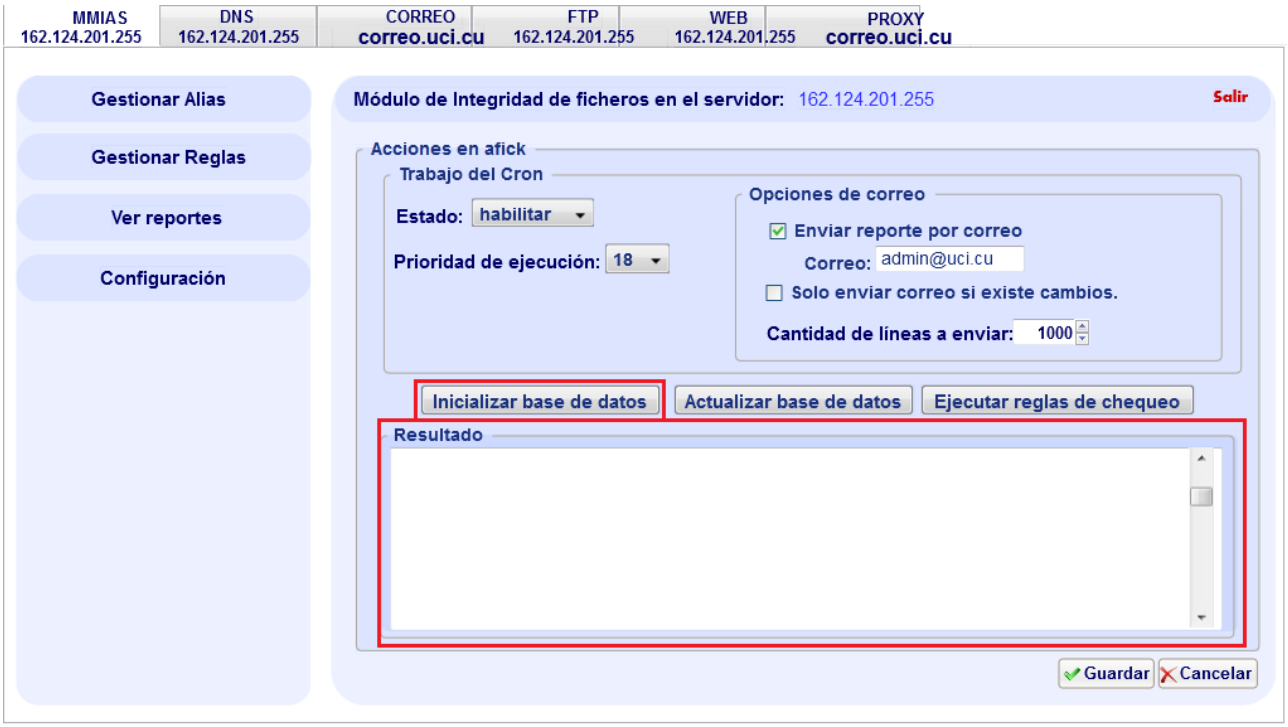
Anexos.

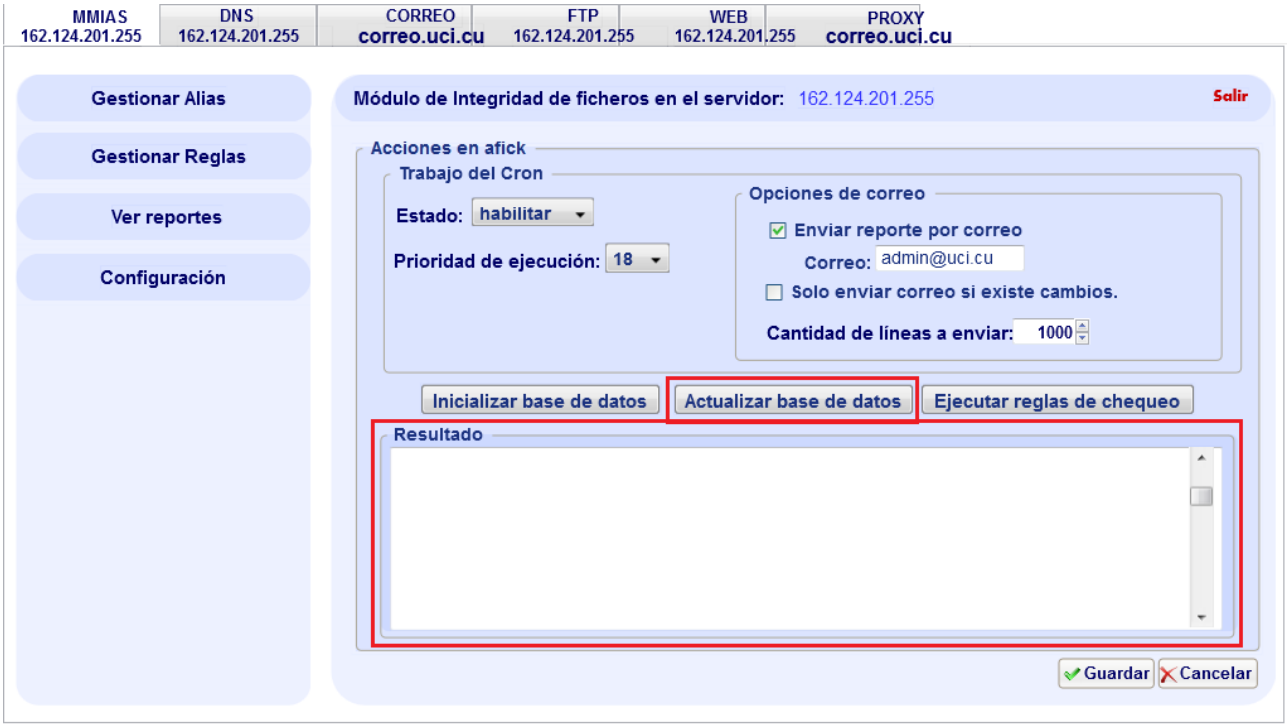
Anexo 1: Asistente de instalación.

Anexo 2: Gestionar alias de reglas de integridad.

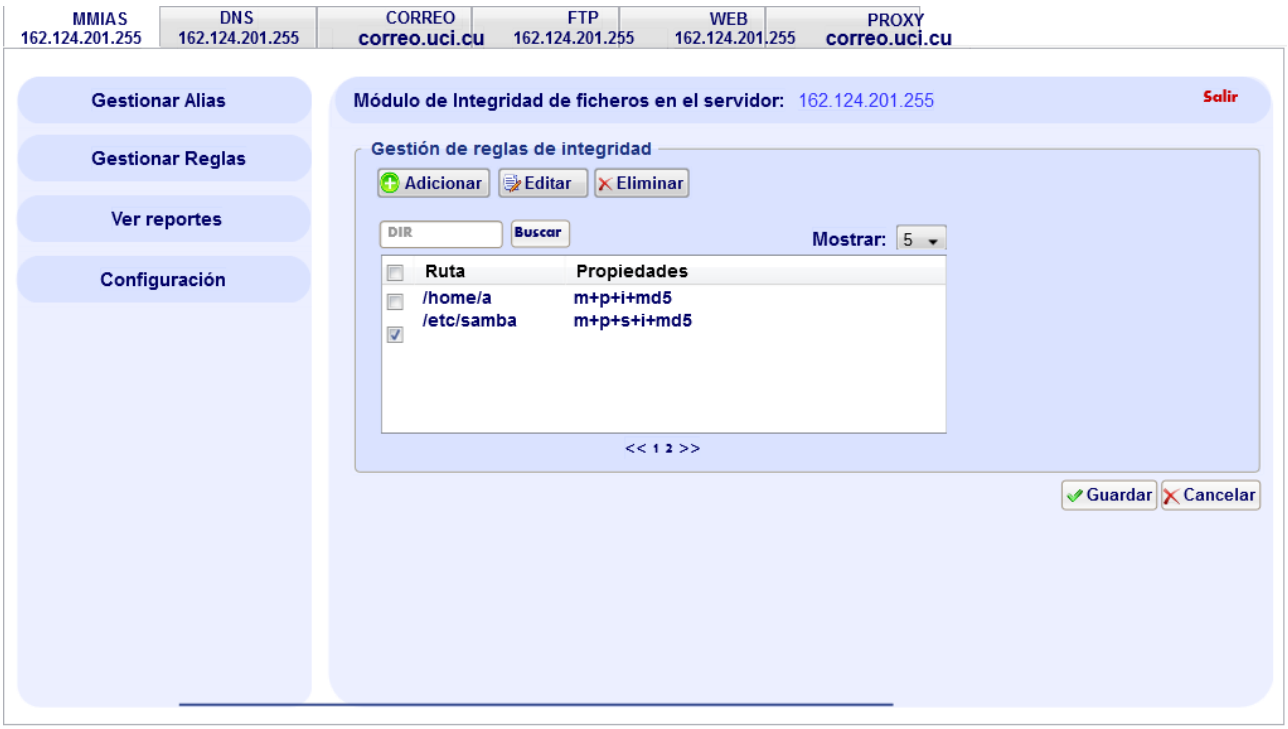


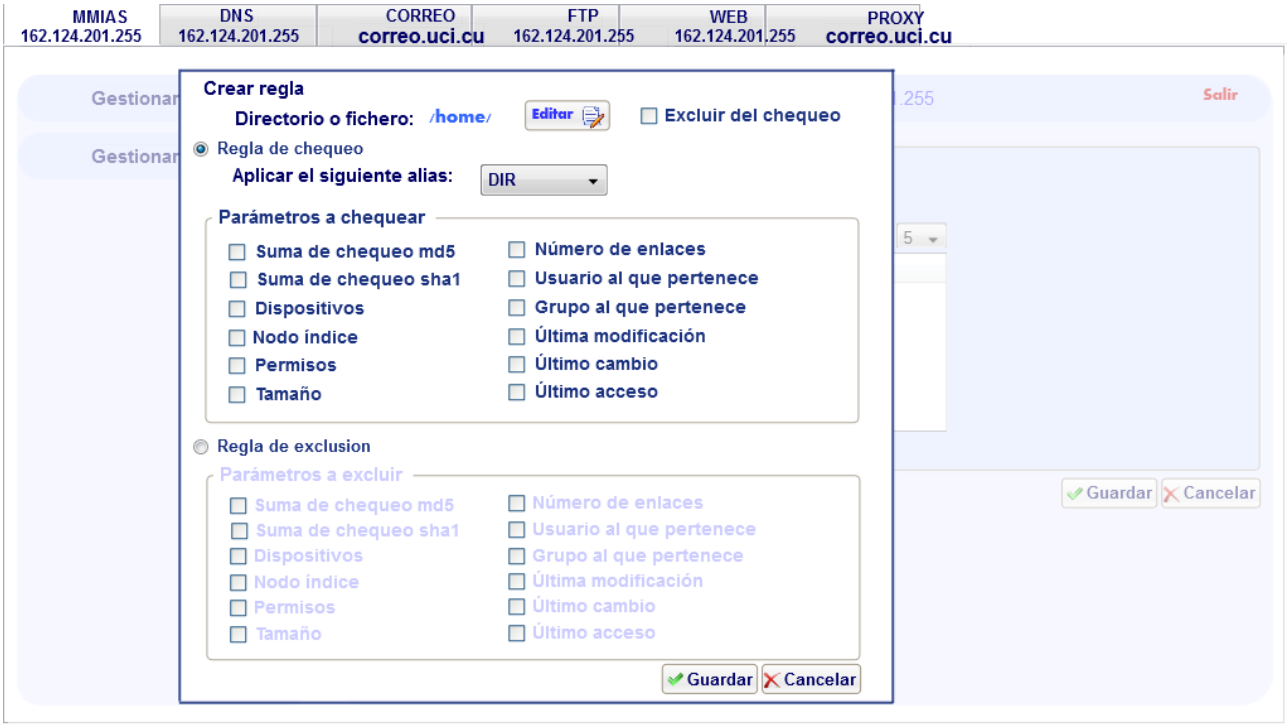
Anexo 3: Gestionar archivo de base de datos.





Anexo 4: Gestionar reglas de integridad.





Anexo 5: Realizar chequeo de integridad.