Ingeniería de Telecomunicación

Escuela Superior de Ingenieros

Universidad de Sevilla

APLICACIÓN PARA LA GENERACIÓN Y ADMINISTRACIÓN DE LA CONFIGURACIÓN DE NAGIOS

Autor: Agustín Bravo Ortiz

Tutora: Isabel Román Martinez

Sevilla, Julio de 2007

Mi más sincero agradecimiento a mi familia, que siempre me ha apoyado y empujado para conseguir llegar hasta aquí, y a mis compañeros de clase, con los que he podido pasar esta carrera y de los que nunca me podré olvidar.

Muchas gracias.

Índice General

PARTE I : INTRODUCCIÓN GENERAL

1 Introducción

1.1 Motivación y Objetivos

En la actualidad, dada la gran cantidad de máquinas y servicios que llegan a componer un entorno de producción y de prestación de servicios a usuarios, se hace indispensable tener instaladas en el mismo herramientas de monitorización de red que se encarguen del chequeo de estas máquinas y servicios. La comprobación de que estos servicios y máquinas están activos y funcionan correctamente llega a ser una labor que no puede llevar a cabo un administrador y que es preferible delegar en un servicio automático o monitor de red. Al mismo tiempo surge un nuevo problema con esta acción, y es que la configuración y mantenimiento de este monitor de red se vuelve cada vez más difícil debido al crecimiento del número de máquinas y servicios. La modificación de los archivos de configuración puede convertirse en una labor imposible y esto nos lleva a una situación parecida a la inicial en la que no comprobamos el estado de las máquinas y los servicios.

El programa Nagios es usado como herramienta de chequeo de servidores y servicios en el servicio de Producción del departamento de Informática en la Consejería de Hacienda. Como se describe anteriormente se llega a un punto en el que su configuración es muy ardua y lenta debido al gran número de páginas de configuración que tienen sus archivos y su difícil comprensión. Por esto se propone la creación de una herramienta que ayude a los usuarios administradores del sistema en la configuración de Nagios y en su actualización sin que conlleve un acceso a los archivos de configuración.

El presente proyecto parte de la necesidad de tener una herramienta de gestión gráfica y de una organización sencilla de la configuración del programa Nagios.

Se busca que la herramienta para la gestión de la configuración pueda permitir las siguientes acciones:

- Visión de la actual configuración del servicio. Se han de ver todos los

parámetros configurados y los que se pueden configurar para un elemento de configuración de Nagios. Todo ello adaptándose a la documentación que la herramienta Nagios tiene para su correcta configuración.

- Almacenamiento de la configuración en una base de datos. Para lo cual se ha de desarrollar esta base de datos en función de los objetos de configuración de Nagios.

- Carga de la configuración de Nagios a la base de datos. Esta utilidad se desarrollaría para usar la actual configuración del sistema sin tener que escribirla a mano en la base de datos que la herramienta de gestión tendría para la configuración de Nagios.

- Desarrollo de la aplicación de forma flexible, permitiendo la posibilidad de añadir nuevas utilidades en la configuración de Nagios para que su adaptación a las siguientes versiones de la herramienta sean fáciles y no conlleven un tiempo excesivo.

- Acceso seguro a la aplicación permitido sólo a los administradores de Nagios.

En el momento en que se decide la preparación de esta herramienta para la gestión de la configuración de Nagios se distribuía la versión 2,6, aunque la versión que se usaba en el sistema era la 1.5. Para que el objetivo de carga de la configuración sea posible se han de efectuar una serie de cambios en la configuración antigua para poder adaptarla a la nueva versión de Nagios.

Para la realización de la herramienta de gestión se tomó como requisito el que fuese una aplicación que siguiese el estandard J2EE. La elección de este tipo de aplicación se toma principalmente por aproximación al resto de aplicaciones generadas en el entorno y para facilitar la reutilización de esquemas de código e instalación de servicios que puedan permitir un avance más rápido del diseño de la aplicación. Relativo al tipo de aplicación está el requisito de que la aplicación tuviese como apoyo el software propietario de Oracle, debido a que en el momento de realización de este proyecto la consejería de

Hacienda tiene un contrato para uso de Software de este distribuidor y por lo tanto los servicios necesarios para la implementación de la aplicación se prefiere que corran sobre los que ya se usan.

En la memoria se detallará en que puntos se ha usado parte de código de otra aplicación o ideas que han sido adoptadas de otras aplicaciones realizadas en el mismo entorno para la generación de la herramienta.

Durante la realización del proyecto se han añadido nuevos requisitos adicionales que se vieron interesantes y que podrían darle un valor añadido a la herramienta, como fueron el que la herramienta pudiese realizar salvaguardias de la configuración para recuperar un punto de restauración anterior o que la herramienta pudiese chequear la configuración generada automáticamente para buscar posibles errores de sintaxis en los datos o verificar que la configuración generada era satisfactoria. Estos puntos serán detallados más adelante en el desarrollo de esta memoria.

1.2 *Fases de Implementación*

Para el desarrollo de este proyecto ha sido necesaria una formación en varios ámbitos para conocer aspectos como el estandard J2EE, diseño de bases de datos, configuración y manejo de Nagios y otros programas que se han usado en el desarrollo de la aplicación. El conjunto de tareas que componen el proyecto se puede dividir en los siguientes apartados:

- Documentación: Fue necesaria una labor de documentación en aspectos como el desarrollo de aplicaciones siguiendo el estandard J2EE, configuración del contenedor Oracle Aplication Server y despliegue de aplicaciones en el mismo, diseño de bases de datos usando la aplicación Oracle Designer y por supuesto un estudio de la configuración que usa la aplicación Nagios para su funcionamiento y verificación de la misma. Una parte fundamental de este proceso de documentación ha sido el estudio del funcionamiento de la aplicación de “Subastas de Liquidez” (aplicación anteriormente realizada para la consejería de Economía y Hacienda), la

cual es mantenida en el departamento de Producción y utiliza un esquema de funcionamiento basado en Servlets el cual es adoptado en su mayor parte por la aplicación de este proyecto.

- Diseño de la base de datos: La base de datos está basada en los objetos de configuración de Nagios y creada a partir de unos diagramas de entidad relación creados con Oracle Designer.

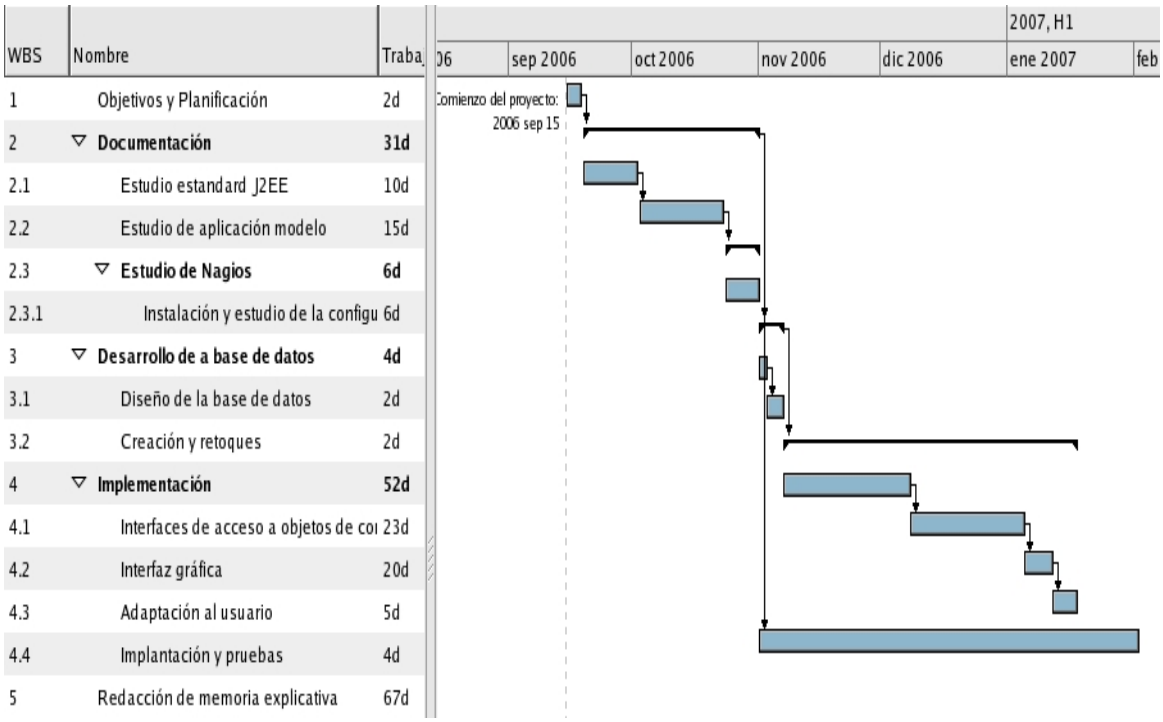
- Creación de los objetos java que emulan los objetos de configuración de Nagios y las clases asociadas a los mismos para su gestión en la base de datos.

- Creación de una interfaz web para la presentación de los parámetros correspondientes a los objetos de configuración y su modificación o inserción.

- Depuración de la aplicación y retoques para adaptarla a un uso real. En esta parte se modificó ligeramente la aplicación cambiando los objetos de configuración para añadirles otras funcionalidades adicionales y también se modificó la interfaz web para restringir ciertos accesos y permitir otras funcionalidades a la interfaz, como la gestión de copias de seguridad de la configuración.

- Redacción de esta memoria en la que se presenta la estructura de la aplicación y la teoría en la que se basa.

En la siguiente figura se presenta mediante un diagrama de Gaant las fases de implementación comentadas anteriormente:



WBS	Nombre	Traba
1	Objetivos y Planificación	2d
2	**Documentación**	**31d**
2.1	Estudio estandard J2EE	10d
2.2	Estudio de aplicación modelo	15d
2.3	**Estudio de Nagios**	**6d**
2.3.1	Instalación y estudio de la configu	6d
3	**Desarrollo de a base de datos**	**4d**
3.1	Diseño de la base de datos	2d
3.2	Creación y retoques	2d
4	**Implementación**	**52d**
4.1	Interfaces de acceso a objetos de cor	23d
4.2	Interfaz gráfica	20d
4.3	Adaptación al usuario	5d
4.4	Implantación y pruebas	4d
5	Redacción de memoria explicativa	67d

Figura 1: Diagrama de Gaant

1.3 *Material y Método*

Para la realización de la aplicación se han usado diversos programas para las tareas de desarrollo, gestión de versiones, despliegue de la aplicación, almacenamiento de datos y pruebas de la aplicación.

La máquina en la que se ha desarrollado la aplicación ha sido un PC con procesador de 2 GHz y 1 GB de RAM con sistema operativo Linux Fedora Core 5.

La herramienta de desarrollo ha sido Eclipse 3.2.1 y el contenedor de pruebas para el despliegue de la aplicación Oracle Aplication Server 10.1.3 en su versión Standalone instalado en la propia máquina. Ha sido de gran utilidad el hecho de que Eclipse se pueda configurar para iniciar el contenedor y desplegar la aplicación automáticamente en el Contenedor Web.

Para la gestión de versiones del programa se ha usado un servidor CVS instalado también en la misma máquina. Se ha configurado eclipse para que haga uso de este servidor de versiones vía ssh y se conecte a él guardando las distintas versiones a medida que se ha ido progresando.

En cuanto a la base de datos usada ha sido una base de datos de Oracle versión 10g instalada en una máquina de Sun con Solaris 8. Ésta era una base de datos para aplicaciones en fase de desarrollo de la Consejería de Economía y Hacienda.

Se ha usado Nagios 2.6 en la cual se ha ido probando la configuración generada por la aplicación y la lectura de la configuración que se usaba.

PARTE II : INTRODUCCIÓN TEÓRICA

2 Sistemas de Monitorización de Red

2.1 Definición

Un sistema de monitorización de red es una herramienta que de forma periódica chequea tanto las máquinas de una red como los servicios que puedan correr sobre ellas. Cuando nos referimos a las máquinas no sólo hablamos de servidores, sino también de switches, balanceadores y toda máquina que pueda responder a peticiones para comprobar que esta activa o peticiones de chequeo de parámetros de rendimiento. Igualmente los servicios (servidores web, bases de datos, proxys ...) podrán chequearse para comprobar que están activos y también que su funcionamiento es el esperado y se podrán configurar chequeos de la parametrización de una máquina para comprobar su ocupación de disco, su carga en memoria o su porcentaje de CPU usado.

Un sistema de monitorización de red tiene la gran utilidad de liberar a un administrador de sistemas de hacer él mismo esos chequeos, y además de notificarle ciertos eventos que se puedan registrar durante los chequeos, como la caída de un servicio o la vuelta al funcionamiento de otro. La forma de chequeo variará de una herramienta a otra. En un caso práctico un administrador podría tardar mucho en llegar a comprobar que un problema ha ocurrido, en el mejor de los casos sería un usuario el que acabaría contactando con él para indicárselo.

Es por esto que, al solucionar un sistema de monitorización de red el problema anterior se diga que gracias al mismo se puede desarrollar una estrategia proactiva ante determinados problemas y no tener que seguir una estrategia reactiva, que sería el caso del usuario que avisa al administrador de sistemas.

2.2 Herramientas de Monitorización

Entre las herramientas de monitorización podremos incluir aquellas que hacen labores menores como las que sólo escanean puertos y tráfico de red. Este es el caso de netoscope, tcpdump, ethereal, etc. Estas herramientas nos permitirán por supuesto

comprobar que una máquina no tiene activo determinado servicio o evidenciar que una máquina esta "caída" porque no obtenemos tráfico que la misma generaría.

Estas herramientas serán fáciles de utilizar y permitirán el análisis de problemas, pero serán poco adaptables a un entorno en el que los chequeos hayan de ser múltiples y se necesite automatización y notificación.

Desde el enfoque de esta introducción, lógicamente veremos que Nagios sí alcanza nuestras expectativas y consigue lo que no obteníamos de las herramientas anteriores, aunque el lector quedaría evidentemente engañado si no se dijese que Nagios no es la salvación, sino que hay otros muchos programas en el mercado que consiguen lo mismo e incluso pueden llegar a cotas superiores. En el mercado podremos encontrar sistemas muy complejos de monitorización como HP openview, Tivoli, etc. Estos sistemas serán aplicaciones muy potentes que en la mayoría de los casos excederán nuestras necesidades, requerirán el pago de licencias y serán poco flexibles.

2.3 Herramientas de Monitorización basadas en Software Libre

En el mercado tendremos herramientas basadas en software libre con las que se conseguirán las expectativas anteriores, posiblemente en algunos casos de forma más simple y con menor categoría en cuanto a presentación y expectativas, pero siempre sin dejar de ser útiles y cumpliendo una necesidad actual como es la distribución de software de fuentes abiertas.

Además, serán herramientas adaptables y modificables para su adaptación a necesidades específicas del usuario y al ser software libre tendremos una fácil integración con otras aplicaciones externas y una sencilla incorporación de scripts.

Estos propósitos son conseguidos por herramientas como Sysmon, Angel Network Monitor, Mars o Nagios.

2.4 Nagios

2.4.1 Introducción a Nagios

Nagios es una herramienta de monitorización de red creada por Ethan Galstad. Como se ha descrito antes puede configurarse para la vigilancia de máquinas, servicios y otros aspectos internos a una máquina o a un servicio que puedan chequearse.

Nagios se encarga del chequeo de estos "hosts" y "services" configurados y simplemente avisa al administrador mediante el tipo de notificación que se configure de que el servidor o servicio ha caído y también de cuando ha mejorado el estado de uno de ellos. Decimos simplemente porque sería un error pensar que Nagios nos va a permitir gestionar una red, no va a ser útil para iniciar o parar servicios o para configurarlos, sólo se encargará del chequeo y del aviso, lo cual es una utilidad que en un ámbito de muchas máquinas y servicios activos llega a ser muy importante.

Nagios está escrito en C y es Software Libre (Licencia GPL Versión 2), lo cual lo convierte en una herramienta muy flexible y adaptable.

2.4.2 Características de Nagios

Nagios nos ofrece múltiples opciones para el chequeo de servicios y para la notificación a sus usuarios. Sus características principales son estas:

- Monitorización de servicios de red: SMTP, POP3, HTTP, DNS, etc.

- Monitorización de recursos de una máquina: carga del procesador, espacio libre en disco, uso actual de la memoria, etc.

- Capacidad de desarrollar plugins de forma sencilla, lo cual permite a los usuarios programar nuevos chequeos según sus necesidades. Los chequeos anteriormente mencionados son ejecutables o scripts con una salida adaptada a Nagios, por lo que igualmente el administrador puede generar nuevos scripts de

chequeo fácilmente.

- Capacidad para definir mediante configuración una topología de red que permita definir qué servicios estarán caídos o serán inaccesibles ante fallo de otros servicios situados por encima de éstos. Esto es una utilidad ante la posibilidad de recibir notificaciones masivas sobre servicios que no es que estén desactivados, sino que son inaccesibles.

- Envío de notificaciones mediante múltiples métodos cuando surgen problemas en los servicios y servidores configurados o cuando éstos son resueltos (vía email, alerta sonora, sms, y los que se puedan configurar).

- Capacidad de definir eventos para que se ejecuten en caso de detectarse problemas, lo cual nos permite definir una estrategia proactiva de control de la red.

- Consola web para la visualización actual del estado de todos los servicios, estadísticas de funcionamiento, historial de alarmas, etc.

2.4.3 Instalación y Configuración de Nagios

La propia documentación de Nagios nos detalla los pasos a seguir para instalar el servicio y nos ofrece una configuración básica para partir con algunos ejemplos de chequeos en la propia máquina.

En la documentación se nos dan algunos consejos para la configuración de Nagios, como son paciencia, una lectura concienzuda de la documentación completa de Nagios y hacer uso inicial de los ficheros de configuración que vienen como ejemplo.

Tras la instalación de Nagios habrá que configurar un servidor Web, como Apache en el caso de sistemas Linux, para que nos pueda mostrar la interfaz web de gestión de Nagios. La configuración del servidor web ha de hacerse para permitir el acceso a la aplicación sólo a determinados usuarios, ya que la información que se muestra y las acciones que permite iniciar la interfaz web no deben estar al alcance de cualquier usuario

de la red.

2.4.4 Objetos de Configuración de Nagios

Nagios divide su configuración en objetos de configuración, los cuales son definidos por la documentación de Nagios como "definiciones de datos que serán útiles para la monitorización de cualquier cosa".

Esta división en objetos de configuración permite implementar la configuración en partes, el administrador del servicio puede ir configurando primero servicios, después servidores, contactos, etc. Por supuesto que estos objetos de configuración estarán interrelacionados y deberemos de tener presentes los demás objetos mientras configuramos uno, pero al estar separados la configuración y corrección de errores de sintaxis en la configuración será más sencilla.

Estos objetos de configuración son (en el mismo orden que presenta la configuración de Nagios) los siguientes:

- Host: Define un servidor físico, una estación de trabajo u otro dispositivo que resida en la red. Estos serán chequeados para comprobar que se encuentran activos.

- Hostgroup: Define un grupo de hosts para poder presentar el estado conjunto y estadísticas de los mismos.

- Service: Identifica un servicio que corre sobre un "host" definido anteriormente. El término servicio en este caso es muy amplio, puesto que define desde un servicio de la máquina (POP, HTTP, SMTP, etc.) hasta comprobaciones a las que pueda responder la máquina (pings, número de usuarios conectados a la máquina, espacio libre en disco,etc.).

- Servicegroup: Define un grupo de servicios para poder presentar estado y estadísticas de los mismos.

- Contact: Define un usuario de Nagios el cual será contactado ante determinados eventos en la red.

- Contactgroup: Define un grupo de "contacts" con el propósito de facilitar la configuración y disponer mediante una sola sentencia que varios contactos sean los referidos ante un evento en la red.

- Time Period: Define una lista de tiempos que serán considerados como válidos para la realización de chequeos o notificaciones a usuarios.

- Commands: Definen comandos que serán ejecutados para la realización de chequeos de servicios o máquinas y también para la notificación a contactos.

- Service Dependencies: Esta característica de Nagios nos permite definir unas dependencias entre servicios para mejorar las notificaciones a los contactos en caso de que caiga un servicio del cual dependan otros. Al ocurrir esto sólo se informará sobre la caída del servicio superior y no se informará sobre los servicios dependientes.

- Service Escalations: Define procedimientos de notificación. Si un servicio o máquina cae durante un tiempo, al pasar de determinado número de notificaciones podremos configurar que se notifique a más contactos de Nagios, propagando así aún más la alarma.

- Host Dependencies: Define dependencias entre "hosts" con el mismo objetivo que para los servicios.

- Host Escalations: Define procedimientos de notificación para los hosts con la misma utilidad que en el caso de los servicios.

- Extended Host Information Definition: Se define información adicional para las máquinas para que Nagios pueda implementar utilidades adicionales al chequeo como la representación gráfica de la arquitectura de la red.

- Extendended Service Information Definition: Información adicional para

las definiciones de servicios.

Para los objetos de configuración Contacts, Services y Hosts la configuración de Nagios permite el uso de plantillas. Esto es, se permite definir objetos con características básicas que después serán usadas por otros objetos como plantillas. Así disminuimos el número de atributos comunes que hay que fijar en un grupo de objetos. Cuando un objeto usa una plantilla se suponen definidos sus parámetros como los de la plantilla, pero en la definición del objeto se podrán redefinir esos atributos sustituyendo con un nuevo valor el que toman de la plantilla.

2.5 Otras Herramientas de Configuración de Nagios

En el mercado del software libre existen actualmente otras herramientas distintas a la que se desarrolla en este proyecto, aunque con características distintas. Estas herramientas pueden ser descargadas desde http://www.nagiosexchange.org/Configuration.20.0.html, página en la que encontramos diversos plugins para Nagios.

Estas herramientas que permiten la configuración de Nagios vía web están desarrolladas en PHP y acceden a bases de de datos MySQL. El estar programadas en PHP tiene el inconveniente de no tener una modularidad y estructuración que tienen otros lenguaes, ya que su código suele estar lleno de “includes” y el seguimiento del programa para un nuevo programador que quiere ampliar la aplicación llegar a ser muy complicado . Es por esto que, este proyecto que desarrolla una primera herramienta en java para la gestión de la configuración, es un avance notable y un punto de partida sobre el que poder seguir trabajando en versiones posteriores adaptándolo a los cambios en la configuración de Nagios.

Un programa desarrollado en java, que use Servlets, cuyas clases estén bien distribuidas y organizadas en paquetes siempre será un programa mucho más fácil de adaptar y modificar.

3 Bases de Datos

Puesto que la gran utilidad de la aplicación sobre la que versa esta memoria es concentrar la configuración de la aplicación Nagios en una base de datos dedicaremos, unos puntos a hablar sobre las utilidades de las bases de datos. Principalmente nos centraremos en las ventajas que conlleva su uso.

3.1 Introducción a las Bases de Datos

Una base de datos es un conjunto de datos que pertenecen al mismo contexto almacenados sistemáticamente para su posterior uso. Estos datos están almacenados de forma coherente y calculada permitiendo una consulta eficiente de los mismos más efectiva. Además presentan cierta redundancia y capacidad para el acceso de distintos usuarios con distintos roles en cuanto a privilegios sobre los datos almacenados en la base de datos.

Podemos resumir las características más beneficiosas de las bases de datos en los siguientes puntos:

- Independencia de datos y tratamiento: Los cambios hechos en los datos no implican cambios en los programas que acceden a los mismos y viceversa.

- Coherencia de resultados: Tenemos menos redundancia de datos y mediante dependencias se evitan inconsistencias entre los datos almacenados dentro de la base de datos. Lógicamente siempre se tendrá cierta redundancia, pero con el uso de bases de datos conseguimos que ésta no sea confusa y equivoca.

- Mejora en la disponibilidad de datos: La base de datos se limita al guardado de datos y es independiente de aplicaciones. Se guardan datos a modo de catálogo y con ciertos permisos de acceso.

- Gran eficiencia en cuanto a gestión y almacenamiento.

- Compartición de datos: Las bases de datos permiten acceso a varios usuarios entre los cuales pueden compartir datos. Además permiten el acceso simultáneo a estos datos y mediante reglas internas de la base de datos se consigue que "cara al usuario" se tenga acceso simultáneo a los datos.

Las bases de datos han tenido su mayor desarrollo en el ámbito informático donde disponemos de Sistemas Gestores de Bases de Datos (SGBD) que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Hoy en día gran parte de las aplicaciones informáticas que hacen uso de una colección de datos hacen uso de Bases de Datos ya que éstas agilizan mucho el manejo de los mismos y proporcionan un rendimiento mayor al programa.

Tendremos una amplia variedad de bases de datos dependiendo del contenido que éstas almacenan y de cómo tratan este contenido.

3.2 Tipos de Bases de Datos

Tomando como criterio la variabilidad de los datos que se guardan en estas bases de datos podremos distinguir las bases de datos entre dos categorías:

- **Bases de datos Estáticas** : Serán aquellas cuyos datos son fijos y están dispuestos con permisos de sólo lectura para el usuario común de la base de datos. Estas bases de datos suelen almacenar datos históricos que lógicamente se suponen fijos. Son datos de consulta.

- **Bases de datos Dinámicas**: Éstas son bases de datos cuyos datos se esperan que sean cambiantes en el tiempo. Sobre éstas se harán no sólo operaciones de consulta, sino también operaciones de actualización o adición de datos nuevos.

Otro criterio que podemos usar para clasificar las bases de datos y distinguirlas entre sí es la naturaleza de los datos que estas guardan. Según este criterio las bases de datos podrían ser: bibliográficas, de texto completo, directorios, bancos multimedia

(imágenes, audio, vídeo, etc.), bibliotecas de información Biológica, etc.

3.3 Modelos de Bases de Datos

Además de clasificar las bases de datos según la variabilidad de sus datos tenemos una clasificación mucho más importante, y es la que distingue las bases de datos según las relaciones que se establecen entre los datos que almacenan. Esta peculiaridad es muy importante en las bases de datos ya que comprende la redundancia que tienen y la consistencia de los datos. Según los distintos modelos que se distinguen actualmente podemos disponer las bases de datos en los siguiente tipos:

- **Bases de datos Jerárquicas**: Estas bases de datos almacenan su información en una estructura jerárquica. Los datos se almacenan jerárquicamente en niveles distintos y dando la sensación de una estructura de árbol en la que un nodo "raíz" tiene uno o varios nodos "hijos" y éstos a su vez pueden ser también nodos "raíz" y tener nuevos "hijos". Este modelo es muy útil para relaciones 1:N donde sólo tenemos relación de nodo raíz (o padre) a nodo hijo, pero no al contrario. La principal deficiencia de este modelo es la de no poder representar la redundancia que sigue la base de datos.

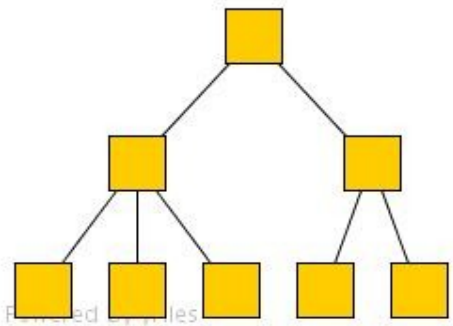


Figura 2: Esquema de una base de datos jerárquica

- **Bases de datos de Red**: Estas base de datos son las que siguen el modelo de red. Este modelo introduce una variación respecto del anterior, y es que los datos ya no siguen unas relaciones de "árbol", ya que ahora un nodo además de poder tener varios nodos "hijo" podrá tener varios nodos "padre". Al introducir esta innovación respecto al modelo anterior ahora sí tenemos una mayor redundancia en los datos y mayor consistencia. El problema que conlleva este modelo es que hace más difícil la administración de la base de datos y el manejo de los datos.

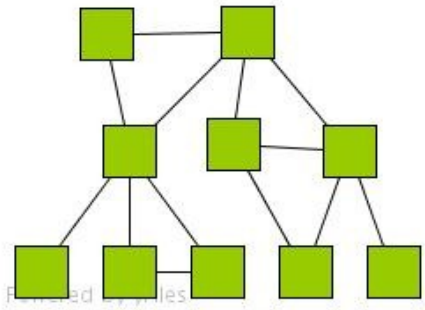


Figura 3: Esquema de una base de datos de red

- **Bases de datos Relacionales**: En 1970 Edgar Frank Codd sentó las bases del modelo Relacional, el cual es actualmente el modelo más usado por las bases de datos. Estas bases son una estructura de datos que sigue la información en la que se tiene una lógica de relaciones basada en tuplas. Una tupla es una relación entre un elemento y varios campos, si unimos varias tuplas lo que tenemos es la base fundamental del modelo relacional, esto es las tablas. Las tablas son conjuntos de filas, cada fila es una tupla con una relación de datos, cada uno correspondiente a un campo (columna).

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario

esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

Además estas tablas que conforman la información en la base de datos se relacionan entre sí. Se establecen relaciones de claves entre columnas de distintas tablas creando una dependencia que será muy útil a la hora de insertar datos y de extraerlos.

- **Bases de datos orientadas a Objeto**: Estas bases de datos son mucho más recientes que el resto y en ellas se intenta asimilar su estructura con la estructura que tienen los lenguajes de programación orientados a objeto. Se intenta que la información se clasifique en objetos independientes los cuales tienen propiedades propias de objetos:

    - Privacidad y encapsulación de datos: Como en objetos de programación se definirán atributos y campos de información que no serán visibles desde el exterior del objeto y que sólo podrán ser usados por él mismo.

    - Herencia: Se establecen jerarquías entre los datos y se podrá tener una herencia de propiedades entre ellos.

    - Comunicación con otros objetos y ejecución de métodos internos que devuelven datos como si se tratase de una consulta a una tabla. Los programas de usuario pueden interactuar con la bases de datos accediendo a ella mediante estos procedimientos, invocándolos con sus parámetros de entrada y recogiendo parámetros de salida.

Estas bases de datos están actualmente en fase de investigación y desarrollo, intentan aunar las características más beneficiosas de las anteriores y añadir nuevas mejoras de control de concurrencia, recuperación, consistencia y lenguaje de consulta.

Entre estos modelos de bases de datos vistos aquí el que se usa para este proyecto

es una base de datos relacional, aunque la base de datos sobre la que se realiza tiene características mucho más avanzadas como la creación de procedimientos internos que manejan los datos antes de devolverlos tras una consulta hecha con lenguaje SQL.

4 Firma Electrónica

Aunque no es un requisito indispensable, dado que el acceso a la aplicación no es crítico para los administradores, se puso como requisito que el acceso a la aplicación se hiciese mediante la autenticación del usuario por su firma electrónica o certificado digital. En este punto exponemos ligeramente en qué consiste este método de autenticación.

4.1 Certificación de la FNMT

La Fábrica Nacional de Moneda y Timbre - Real Casa de la Moneda (FNMT-RCM), como Proveedor de Servicios de Certificación a través de CERES (CERtificación ESpañola), ha implementado una serie de aplicaciones que permiten a la Administración, a los ciudadanos y a las empresas españolas realizar sus trámites a través de Internet de forma totalmente segura. Las nuevas soluciones de certificación y autentificación de identidad digital que ofrece la FNMT-RCM proporcionan validez y seguridad a las transacciones electrónicas.

Entre los medios que dispone la FNMT-RCM para los usuarios se encuentra el sistema de certificación de firma electrónica, el cual es un intermediario transparente al ciudadano, de fácil uso, que ofrece alta disponibilidad y gran capacidad de acceso concurrente para los usuarios de la red. Este sistema ya está operativo con diversas aplicaciones en funcionamiento y numerosos organismos en fase de incorporación al sistema. Los certificados emitidos por la FNMT-RCM son de uso general y por lo tanto universales, es decir, cada ciudadano puede comunicarse con las diferentes administraciones con un único certificado.

4.2 CERES

El proyecto CERES (CErtificación Española) liderado por la FNMT consite, en lineas generales, en establecer una Entidad Pública de Certificación, que permita autentificar y garantizar la confidencialidad de las comunicaciones entre ciudadanos, empresas u otras instituciones y administraciones públicas a través de las redes abiertas de comunicación.

El objetivo principal de CERES hacer seguras las comunicaciones electrónicas con la Administración, siendo un intermediario transparente al usuario que garantizará a ciudadanos y Administraciones la identidad de ambos partícipes en una comunicación, así como la confidencialidad e integridad del mensaje enviado.

Para ello, CERES utiliza técnicas y sistemas criptográficos basados en lo que se conoce como sistema de clave pública, con dos características básicas:

- La identidad del usuario, al igual que su capacidad de firma, se encuentra, en el caso de máxima seguridad, almacenada en una tarjeta inteligente, que no puede ser accesible salvo por su propietario cuando introduzca el número de identificación personal, similar a la clave de una tarjeta de crédito. En caso de no utilizar tarjeta, el perfil criptográfico queda almacenado en un fichero, siendo necesario también un PIN de acceso.

- El sistema es completamente transparente al usuario, es decir, no es necesario conocer ninguna técnica criptográfica para realizar o verificar una firma electrónica o cifrar o descifrar un mensaje.

4.3 Uso del Certificado Digital

Desde el punto de vista del usuario, como se comentaba en el apartado anterior, su certificado digital podrá tenerlo almacenado en una tarjeta o mediante un archivo. El uso que hace del mismo ha de ser personal ya que lo identifica en accesos por Internet a portales de la Administración pública y le permite realizar operaciones como consultar sus

multas de circulación, realizar la declaración de la renta, obtener certificados de vida laboral, de empadronamiento, generación de documentos con validez legal, etc.

Cuando un usuario solicita un certificado, su navegador genera un par de claves. La clave privada se guarda en su navegador y la clave pública se envía a la FNMT-RCM. La FNMT-RCM asignará un código de solicitud a esa clave que le será remitido vía web. Entonces deberá personarse en una oficina de acreditación con su documento de identidad y dicho código. Finalmente, tras la acreditación, podrá proceder a la descarga del certificado vía web. Este quedará instalado en su navegador.

Una vez que el usuario tiene el certificado en su navegador también podrá exportarlo e importarlo en otros navegadores. Esta acción requerirá una contraseña para que no la pueda hacer cualquier persona. La gran mayoría de los navegadores que se ofrecen en el mercado pueden gestionar los certificados de usuario e incluso gestionar varios al mismo tiempo dando la posibilidad a la persona que navega de indicar con qué certificado se quiere autenticar en la aplicación web a la que va a acceder.

La validación del usuario se produce al tomar el servidor de la aplicación el certificado del usuario y validarlo con uno de los servidores de CERES, los cuales informan a la aplicación de que el certificado es válido y de que la información personal del usuario es correcta.

5 Aplicaciones J2EE

La aplicación desarrollada es una aplicación que sigue el esquema de aplicaciones java J2EE (Java 2 Platform, Enterprise Edition). Este estándar describe aplicaciones estructuradas en una serie de capas diferenciadas según funciones y basadas, como indica su nombre, en lenguaje Java.

5.1 Aplicaciones distribuidas en Capas

La lógica de las aplicaciones está divida en componentes acordes con la función que desempeñan. Así mismo estos componentes podrán estar instanciados en distintas máquinas según convenga para la aplicación, ya que entre ellos habrá una comunicación y no es necesario que coexistan todos en la misma máquina. En el siguiente esquema se detalla el conjunto de componentes que llegan a formar la aplicación, aunque no es necesario que existan todos ellos forzosamente.

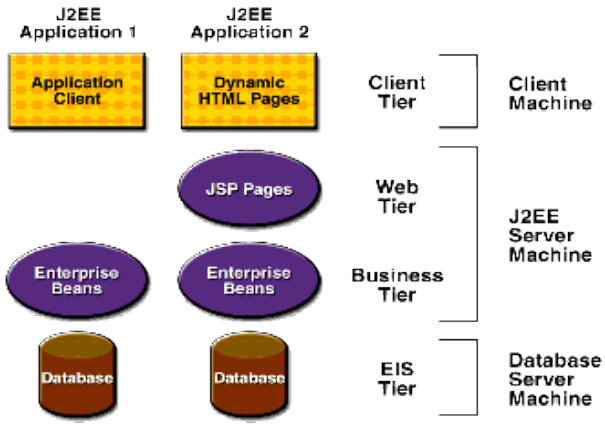


Figura 4: Esquema de capas de una aplicación J2EE

Las capas que se diferencian en la imagen son:

- Capa de cliente: en la que distinguimos las páginas html que acaba

visualizando el navegador del cliente como fruto de todo el proceso de la aplicación.

- Capa Web: Este módulo lo conforman las páginas jsp que se procesan en la máquina del servidor. Estas páginas serán páginas con código html y parte de código java que se ejecuta en el servidor y como resultado se envía una página al cliente de la aplicación.

- Capa de Negocio: En esta capa tenemos los Beans o clases que se encargan del manejo de datos, cálculos con los mismos y acceso a bases de datos para su obtención o almacenamiento. Estas clases gestionan los datos y se encargan de pasar el resultado a la capa superior, a la capa Web, que es la que los presenta al cliente.

- Capa EIS (Enterprise information system): Esta capa viene a ser el almacén de datos de la aplicación, la base de datos a la que se accede en modo consulta o para inserción de datos.

Entre cada una de estas capas se establece un flujo de información iniciado por el cliente. En algunos casos no será necesario que lleguen a interactuar todas las capas para atender una petición ya que podrá ocurrir que simplemente se haga una petición de una página html para la cual no hará falta que la aplicación presente datos extraídos de la base de datos.

5.2 Componentes de una Aplicación J2EE

Tendremos distintos componentes que conforman una aplicación J2EE y que se distribuyen entre las distintas capas explicadas en el apartado anterior. Los componentes que se distinguen en una aplicación J2EE se dividen en tres grupos:

- Clientes de aplicación y Applets que corren en el cliente.

- Servlets y páginas JSP que corren en el lado del servidor.

- Enterprise Beans, que serán las clases que conforman la capa de negocio, las cuales también se ejecutarán en el servidor de la aplicación.

Al igual que no todas las capas pueden llegar a ser necesarias y que no todas han de activarse ante determinadas peticiones del cliente, con los componentes tenemos que no todos serán necesarios y que algunos podrán sustituirse en el conjunto de la aplicación.

5.2.1 Clientes J2EE

Los clientes de aplicaciones J2EE pueden ser tanto clientes Web como aplicaciones que corren en la máquina cliente y se comunican con el servidor sin una interfaz web.

Los clientes Web podemos decir que se dividen en dos partes, una la conforman las páginas "jsp" dinámicas que son generadas en la capa Web del servidor de aplicación y otra el navegador web del cliente que muestra las páginas al usuario. A los clientes Web se les suele llamar clientes ligeros porque acceden a la aplicación a través de la capa Web y no directamente a la Capa de negocio con la base de datos, así mismo muchas de las operaciones que requieran no tendrán necesidad de consultas a la base de datos.

Los clientes Web pueden hacer uso de Applets, las cuales son aplicaciones cliente ligeras que se incluyen en las páginas dinámicas que consulta el usuario cliente. Estas Applets están escritas en java y son ejecutadas por la máquina virtual java instalada en el navegador del cliente. Estas Applets por lo tanto requieren que el navegador del usuario esté adaptado para su uso y serán ejecuciones en la máquina del usuario que pueden ser un agujero de seguridad para el mismo.

La otra modalidad de clientes de las aplicaciones J2EE son aplicaciones java que corren en la máquina del cliente de la aplicación y proveen una forma de acceso al servicio distinta de la que ofrece un navegador Web. Estas aplicaciones normalmente también

dispondrán de una interfaz gráfica de uso y ofrecerán una mayor potencia de actuación con la aplicación. Incluso podría ofrecer una linea de comando al usuario para que éste actuase según su conveniencia. La lógica de estos clientes es que se comuniquen directamente con la Capa de Negocio de la aplicación y que consulten los datos directamente sin hacer uso de capas superiores.

Entre los dos clientes expuestos es más común el uso de clientes Web, ya que no implican una ejecución de aplicaciones en la máquina del usuario cliente y permiten una mayor modularidad de la aplicación y flexibilidad en cuanto a la forma de acceso. Por lo mismo se preferirá también que no incluyan Applets para no depender de requisitos en el navegador del cliente. En la aplicación "nagiosadm" se ha preferido tener un interfaz Web del cliente y tener páginas "jsp" en las que el código java sea lo más reducido posible teniendo la mayor parte del proceso de datos en la Capa de Negocio y usando sólo código java en las páginas jsp para la muestra de datos.

5.2.2 Componentes Web

Los componentes Web de una aplicación J2EE serán servlets o páginas jsp. Los servlets serán clases java que procesan dinámicamente peticiones y respuestas generando páginas Web. Paralelamente se podrán usar páginas jsp, las cuales tienen código html y código java que se procesa antes de construir la página.

La diferencia entre ambos métodos de proceder es que se preferirán los servlets cuando tengamos un procesado mayor de datos y las páginas jsp cuando lo que queramos es incluir pequeñas sentencias java dentro de una página prácticamente estática. Las páginas html y los Applets no serán considerados como partes de una estructura Web de una aplicación J2EE.

En la aplicación que se ha desarrollado se ha preferido no hacer uso de servlets que construyen páginas dinámicas ya que estos suelen tener un código muy engorroso de seguir. Estos suelen tener muchas escrituras de salida formando el código html de una

páginas estática que sería mostrada por el navegador del cliente y el resultado es un código complejo y difícil de administrar. Se ha preferido una comunicación directa de las clases que procesan los datos con las páginas jsp, las cuales se han forzado al máximo a que tengan la menor cantidad de código posible, sólo el necesario para mostrar los datos que se envían al cliente.

5.2.3 Componentes de la Capa de Negocio

Los JavaBeans y EnterpriseBeans compondrán la Capa de Negocio de la aplicación J2EE. Los JavaBeans se encargarán de las consultas, inserciones y actualizaciones de datos haciendo uso del Sistema Gestor de Base de datos y el procesado de estos datos, ya sea antes de la inserción o después de la extracción.

En la siguiente figura se muestra la arquitectura de comunicación completa en la que cada componente se comunica con los adyacentes. Tenemos una estructura modular que nos permite independencia entre capas y una depuración y solución de problemas más sencilla.

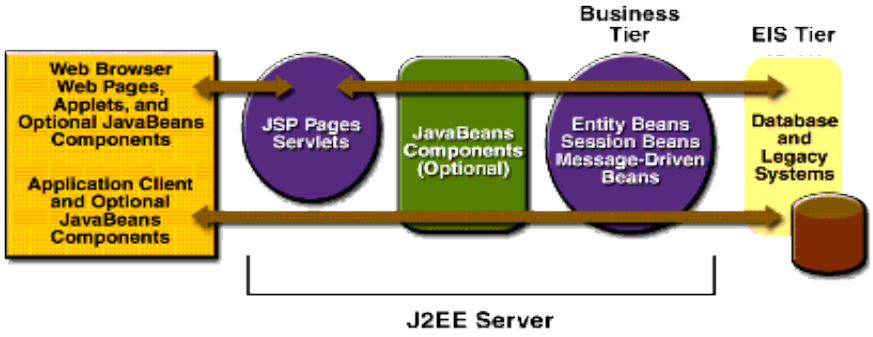


Figura 5: Esquema de comunicación entre componentes de una aplicación J2EE

En la documentación propia del estandard J2EE se distinguen las EnterpriseBeans en tres tipos distintos según el manejo de datos que realizan: las Session Beans tratan los

datos de una conversación de un cliente, cuando ésta finaliza los datos son borrados de memoria. Las Entity Beans son usadas para datos constantes que son almacenados en una tabla, estos datos se han de conservar aunque una sesión con la aplicación finalice. Por último, los Message-Driven Beans aúnan características de los Session Beans y JMS (Java Message Driven), el cual es un API de Java que permite mensajes estándar entre componentes de una aplicación.

PARTE III : IMPLEMENTACIÓN PRÁCTICA

6 Tecnología y Herramientas usadas

6.1 Introducción

En este apartado se comentan las herramientas utilizadas para la realización del proyecto. En la medida de lo posible se han intentado utilizar herramientas de software libre, pero también las circunstancias de realización del proyecto han conllevado que se usen otras herramientas con licencia de uso. El uso de estas herramientas "propietarias" viene impuesto por el usuario final de la aplicación.

6.2 Lenguaje Java y Aplicaciones J2EE

Uno de los prerrequisitos de la aplicación era que estuviese realizada en Java, particularmente que usase el estandard J2EE (Java 2 Enterprise Edition), y así siguiese el patrón de aplicaciones realizadas en el departamento de desarrollo de la Consejería de Hacienda. Así se tendría una mayor seguridad en la aplicación y podría ser alojada con el resto de las aplicaciones más fácilmente.

Java es un lenguaje orientado a objetos, lo cual nos permite una mejor distribución del programa con claridad y una visión de elementos separados que nos beneficiará a la hora de posibles modificaciones o mejoras.

Como entorno de desarrollo java se ha usado la aplicación Eclipse en su distribución para sistema operativo Linux.

6.3 Servidor de Aplicaciones OC4J

Para el alojamiento de la aplicación y la realización de pruebas se ha usado el contenedor de aplicaciones de Oracle, el cual recibe el nombre de OC4J. Se ha usado la versión Standalone 10.1.3 para una instalación local en ámbito de desarrollo y posteriormente se han usado instalaciones de OC4J existentes en otras máquinas para los

entornos de pruebas y de explotación.

Se configuró el entorno de desarrollo, Eclipse, para que automáticamente generase el archivo ".war" (Web Archive) de la aplicación, lo desplegase en el contenedor de aplicaciones y lo arrancase o lo actualizase para que volviese a desplegar la aplicación en caso de que hubiésemos hecho cambios. Así se hicieron mucho más fáciles las tareas de cambios y pruebas.

6.4 Base de Datos Oracle

Para el almacenamiento de la información que maneja la aplicación relativa a la configuración que se genera para Nagios se ha usado una base de datos Oracle versión 10g. Esta base de datos estaba instalada en una máquina de desarrollo y se configuró el Contenedor de aplicaciones para que se conectase a ella. Para ello se creaba una configuración de un pool de conexiones a la base de datos el cual posteriormente era invocado desde la aplicación. Por lo tanto la gestión de usuario y contraseña de conexión a la base de datos no se hacía dentro del código de la aplicación sino en la configuración del OC4J mientras se configuraba el pool de conexiones a la base de datos.

Tras configurar el pool de conexiones, éste se ponía como disponible mediante un recurso JNDI y éste era posteriormente invocado desde la aplicación.

Para el diseño de la base de datos se usado Oracle Designer, el cual a partir de unos diagramas de entidad-relación crea la estructura de tablas de la base de datos. Lógicamente después hubo que hacer algunas modificaciones sobre lo creado automáticamente por la aplicación, ya que algunas claves foráneas no servían o eran inapropiadas.

6.5 Otras Herramientas

Las anteriores herramientas han sido las principalmente usadas para el desarrollo de la aplicación. Lógicamente, aunque no se ha mencionado, se usó una instalación de

Nagios 2.6 sobre la que, en la última fase de pruebas, se iban dejando los archivos de configuración y se chequeaba que se generaban correctamente.

Otras herramientas usadas durante la realización del proyecto han sido para el modelado de la estructura de la aplicación en diagramas UML, para la que se ha usado yed y Visual Paradigm, la realización de esta memoria, con OpenOffice, un servidor CVS sobre Linux, Imendio Planner para la gestión del proyecto.

7 Implementación

En este capítulo de la memoria se irán describiendo los pasos dados para la realización de la interfaz comenzando por el modelado de la base de datos que debía usar y por el modelado de la aplicación según las utilidades que se requerían de la misma. Se comentarán un diagrama de clases de uso y diagramas de clases de la aplicación.

También se comentarán puntos que el autor considera interesantes sobre el desarrollo de la aplicación por haber sido nuevos en cuanto a la introducción a la programación de aplicaciones J2EE como han sido, el sistema de log de la aplicación o la ejecución de una llamada a la aplicación para la realización de una acción concreta.

7.1 Modelado de la Base de Datos

Para el modelado de la base de datos, como se comentó anteriormente, se usó Oracle Designer para crear el primer diagrama de entidad-relación. Este diagrama se creó teniendo en cuenta los objetos de configuración de Nagios y los usuarios que accederían a la aplicación. El resultado de pasar este diagrama a la base de datos fue posteriormente modificado y adaptado.

Para normalizar los nombres de las tablas se dispuso que todas empezasen por “na_”, así se sabría a nivel global que las tablas pertenecían al esquema nagiosadm. En la

siguiente figura tenemos un diagrama de entidad-relación que ayudó a confeccionar la lista de tablas que después se usarían en el proyecto. En el diagrama además de los objetos de configuración también se han incluido algunas funciones para orientarnos en el funcionamiento de Nagios.

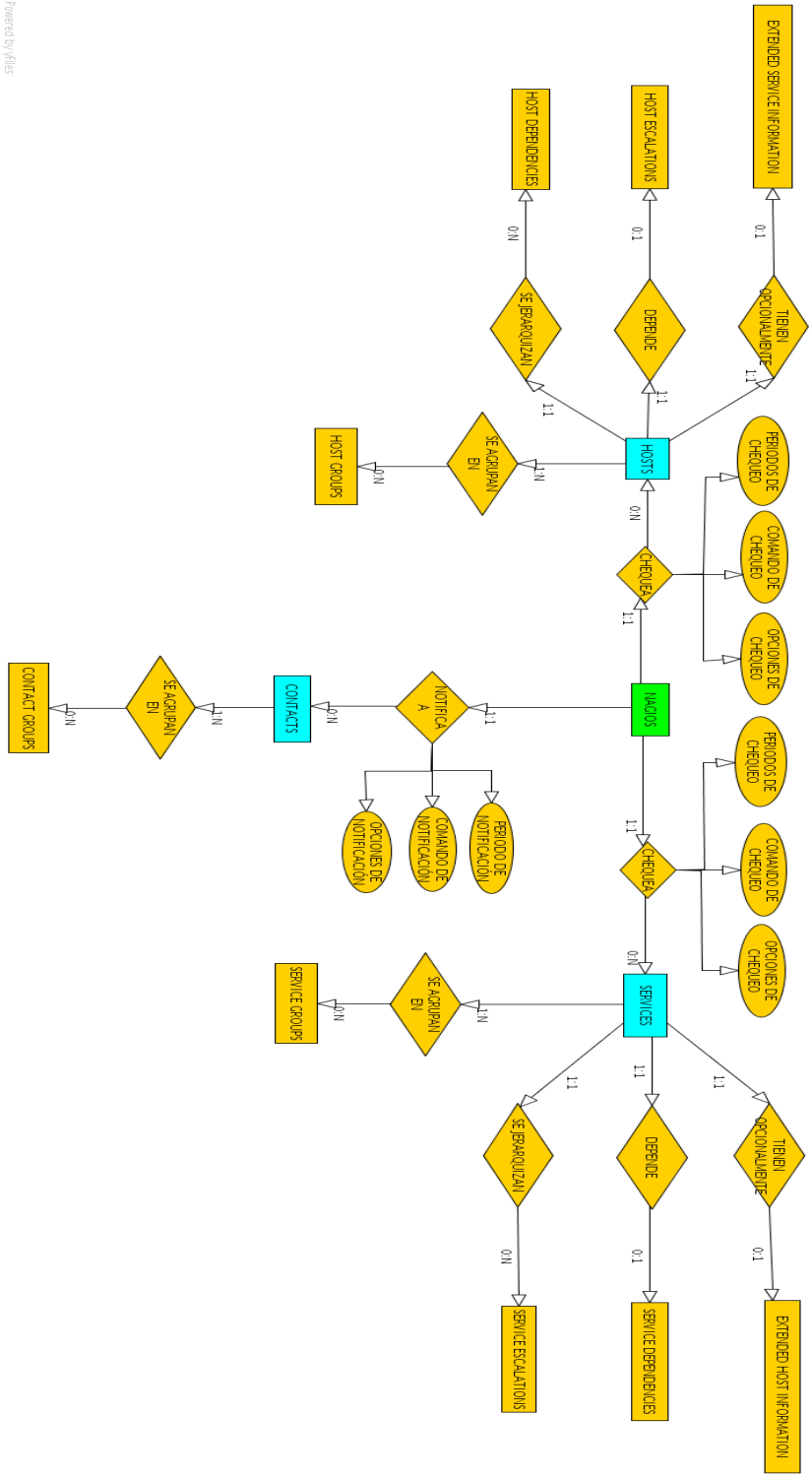


Figura 6: Diagrama Entidad Relación de la base de datos

A partir del resultado inicial de la conversión a tablas del diagrama de entidad relación creado en Oracle Designer y varias modificaciones para establecer relaciones creamos la siguiente lista de tablas. En la explicación de las tablas marcaremos la clave primaria de cada tabla y se indicará si referencia a un registro de otra tabla.

Tabla na_admins: Datos de los usuarios administradores que acceden a la aplicación.

Atributos	**Tipo SQL**
nif	VARCHAR
nombre	VARCHAR
password	VARCHAR
estado	INT
fecude	VARCHAR
email	VARCHAR
admin	INT

Tabla 1: na_admins

Tabla na_commands: Commandos de notificación y chequeo de Nagios.

Atributos	**Tipo SQL**
commname	VARCHAR
commline	VARCHAR
notifica	INT

Tabla 2: na_commands

Tabla na_timeperds: Intervalos de notificación de Nagios con sus atributos.

Atributos	Tipo SQL
timename	VARCHAR
monday	VARCHAR
tuesday	VARCHAR
wednesday	VARCHAR
thursday	VARCHAR
friday	VARCHAR
saturday	VARCHAR
sunday	VARCHAR
alias	VARCHAR

Tabla 3: na_timeperds

Tabla na_platform: Plataformas posibles de clasificación de los elementos de chqueo.

Atributos	Tipo SQL
platform	VARCHAR

Tabla 4: na_platform

Tabla na_contacts: Datos básicos de los contactos de Nagios.

Atributos	Tipo SQL	Claves Externas
id_contact	VARCHAR	
platform	VARCHAR	na_platform.platform
alias	VARCHAR	
name	VARCHAR	
contname	VARCHAR	
template	VARCHAR	na_contacts.name

Tabla 5: na_contacts

Tabla na_typecnats: Tipos de atributos posibles para un contacto.

Atributos	Tipo SQL
typename	VARCHAR
extended	INT
obligato	INT

Tabla 6: na_typecnats

Tabla na_contattrs: Atributos de los contactos.

Atributos	Tipo SQL	Claves Externas
typename	VARCHAR	na_typecnats.typename
id_contact	VARCHAR	na_contacts.id_contact
value	VARCHAR	
ovrd	INT	

Tabla 7: na_contattrs

Tabla na_contgrps: Definiciones de los grupos de contactos de Nagios

Atributos	Tipo SQL	Claves Externas
contgrp	VARCHAR	
alias	VARCHAR	
platform	VARCHAR	na_platforms.platform

Tabla 8: na_contgrps

Tabla na_cgmembers: Relación de los contactos de cada grupo de contactos.

Atributos	Tipo SQL	Claves Externas
contgrp	VARCHAR	na_contgrps.contgrp
id_contact	VARCHAR	na_contacts.id_contact

Tabla 9: na_cgmembers

Tabla na_hosts: Definiciones de los Hosts de Nagios con sus atributos básicos.

Atributos	Tipo SQL	Claves Externas
id_host	VARCHAR	
name	VARCHAR	
platform	VARCHAR	na_platforms.platform
hostname	VARCHAR	
alias	VARCHAR	
template	VARCHAR	na_hosts.name
addrs	VARCHAR	

Tabla 10: na_hosts

Tabla na_typehoats: Tipos de atributos posibles para un host.

Atributos	Tipo SQL
typename	VARCHAR
extended	INT
obligato	INT

Tabla 11: na_typehoats

Tabla na_hostattrs: Atributos de los hosts.

Atributos	Tipo SQL	Claves Externas
typename	VARCHAR	na_typehoats.typename
id_host	VARCHAR	na_hosts.id_host
value	VARCHAR	
ovrd	INT	

Tabla 12: na_hostattrs

Tabla na_hostgrps: Definiciones de los grupos de hosts.

Atributos	Tipo SQL	Claves Externas
hostgrp	VARCHAR	
alias	VARCHAR	
platform	VARCHAR	na_platforms.platform

Tabla 13: na_hostgrps

Tabla na_hgmembers: Relación de los hosts de cada grupo de hosts.

Atributos	Tipo SQL	Claves Externas
hostgrp	VARCHAR	na_hostgrps.hostgrp
id_host	VARCHAR	na_hosts.id_host

Tabla 14: na_hgmembers

Tabla na_services: Definiciones de los Services de Nagios con sus atributos básicos.

Atributos	Tipo SQL	Claves Externas
id_srv	VARCHAR	
name	VARCHAR	
platform	VARCHAR	na_platforms.platform
hostname	VARCHAR	
alias	VARCHAR	
template	VARCHAR	na_services.name
descr	VARCHAR	

Tabla 15: na_services

Tabla na_typesrvats: Tipos de atributos posibles para un Service.

Atributos	Tipo SQL
typename	VARCHAR
extended	INT
obligato	INT

Tabla 16: na_typesrvats

Tabla na_srvattrs: Atributos de los Services.

Atributos	Tipo SQL	Claves Externas
typename	VARCHAR	na_typesrvats.typename
id_srv	VARCHAR	na_services.id_srv
value	VARCHAR	
ovrd	INT	

Tabla 17: na_srvattrs

Tabla na_srvgrps: Definiciones de los grupos de Services.

Atributos	Tipo SQL	Claves Externas
sgname	VARCHAR	
alias	VARCHAR	
platform	VARCHAR	na_platforms.platform

Tabla 18: na_srvgrps

Tabla na_sgmembers: Relación de los Services de cada grupo de Services.

Atributos	Tipo SQL	Claves Externas
sgname	VARCHAR	na_srvgrpssgname
id_srv	VARCHAR	na_services.id_srv

Tabla 19: na_sgmembers

Tabla na_hostescals: Atributos de los objetos de configuración Host Escalations.

Atributos	Tipo SQL	Claves Externas
id_hostescal	VARCHAR	
hostgrp	VARCHAR	na_hostgrps.hostgrp
escalper	VARCHAR	na_timeperds.timename
hostname	VARCHAR	na_hosts.id_host
escalopt	VARCHAR	
lastnotif	VARCHAR	
notifint	VARCHAR	
contgrp	VARCHAR	na_contgrps.contgrp
firstnotif	VARCHAR	

Tabla 20: na_hostescals

Tabla na_hostdepens: Atributos de los objetos de configuración Host Dependencies.

Atributos	Tipo SQL	Claves Externas
id_hostdepen	VARCHAR	
exec_crit	VARCHAR	
notf_crit	VARCHAR	
hostname	VARCHAR	na_hosts.id_host
depenhost	VARCHAR	na_hosts.id_host
inherit	INT	

Tabla 21: na_hostdepens

Tabla na_srvescals: Atributos de los objetos de configuración Service Escalations.

Atributos	Tipo SQL	Claves Externas
id_srvescal	VARCHAR	
srvdescr	VARCHAR	na_services.descr
hostname	VARCHAR	na_hosts.id_host
lastnotif	VARCHAR	
escalopt	VARCHAR	
contgrp	VARCHAR	na_contgrps.contgrp
firstnotif	VARCHAR	
escalper	VARCHAR	na_timeperds.timename

Tabla 22: na_srvescals

Tabla na_srvdepens: Atributos de los objetos de configuración Service dependencies.

Atributos	Tipo SQL	Claves Externas
id_srvdepen	VARCHAR	
hostname	VARCHAR	na_hosts.id_host
depensrv	VARCHAR	na_services.id_srv
depenhost	VARCHAR	na_hosts.id_host
notf_crit	VARCHAR	
inherit	INT	
srvdescr	VARCHAR	na_services.descr
exec_crit	VARCHAR	

Tabla 23: na_srvdepens

Los principales objetos de configuración son los Contacts, Hosts y Services, estos además son los que más atributos tienen y además para alguno de estos atributos se podían

tener varios valores. Por lo tanto se eligió tener identificadores de estos objetos en unas tablas y tener sus atributos en otras tablas siendo referenciados con los identificadores de cada objeto. Durante la instalación de la aplicación se crean las tablas na_typecnats, na_typesrvats y na_typehoats y se rellenan definiendo los posibles atributos que pueden tomar los objetos de configuración Contacts, Services y Hosts respectivamente. Además en estas tablas indicamos en cada registro si el atributo correspondiente es obligatorio en la configuración de Nagios o no, con lo que podemos hacer la comprobación de las inserciones y modificaciones de la configuración que hace el usuario.

En las tablas en las que se insertan los atributos de los Contacts, Services y Hosts, se ha añadido una columna "ovrd" en la que informamos si el atributo que tiene el objeto sobrescribe el valor de la plantilla que use dicho objeto (si es que usa alguna).

Se puede observar que para la gran mayoría de campos se han usado tipos VARCHAR y que en pocos hemos usado INT. Estos últimos se han usado principalmente como tipos booleanos en los que afirmábamos o negábamos algo con valor 1 o 0 respectivamente.

Como último comentario respecto al diseño de la base de datos diremos que las tablas anteriormente expuestas son para el caso en que usamos una base de datos Oracle y que cuando se hizo el paso a MySQL fue necesario añadir unas tablas de apoyo para la generación de secuencias para los identificadores de objeto, acción que en Oracle era más sencilla porque se usaban unos objetos que ya funcionaban como secuencia y estaban configurados en la base de datos. En cuanto a la tabla de usuarios administradores tenemos un campo de password que no se usa en el caso en que autenticamos al usuario con su certificado digital (solo necesitamos que esté registrado en la tabla con su DNI).

Durante la realización de la aplicación se usó el programa "Oracle SQL Developer" para realizar consultas directas a la base de datos. Con la información de debug de la aplicación podíamos ver las consultas que se realizaban a la base de datos y después realizarlas con este programa para ver si las consultas eran correctas, ver si los resultados eran los esperados y que la aplicación los trataba correctamente.

7.2 Modelado de funciones de la aplicación

Inicialmente el propósito de la aplicación era la gestión de la configuración de Nagios, aunque posteriormente se incorporaron otras posibilidades para facilitar el uso a los administradores.

En el siguiente diagrama de casos de uso se pueden ver las interacciones de un usuario con la aplicación:

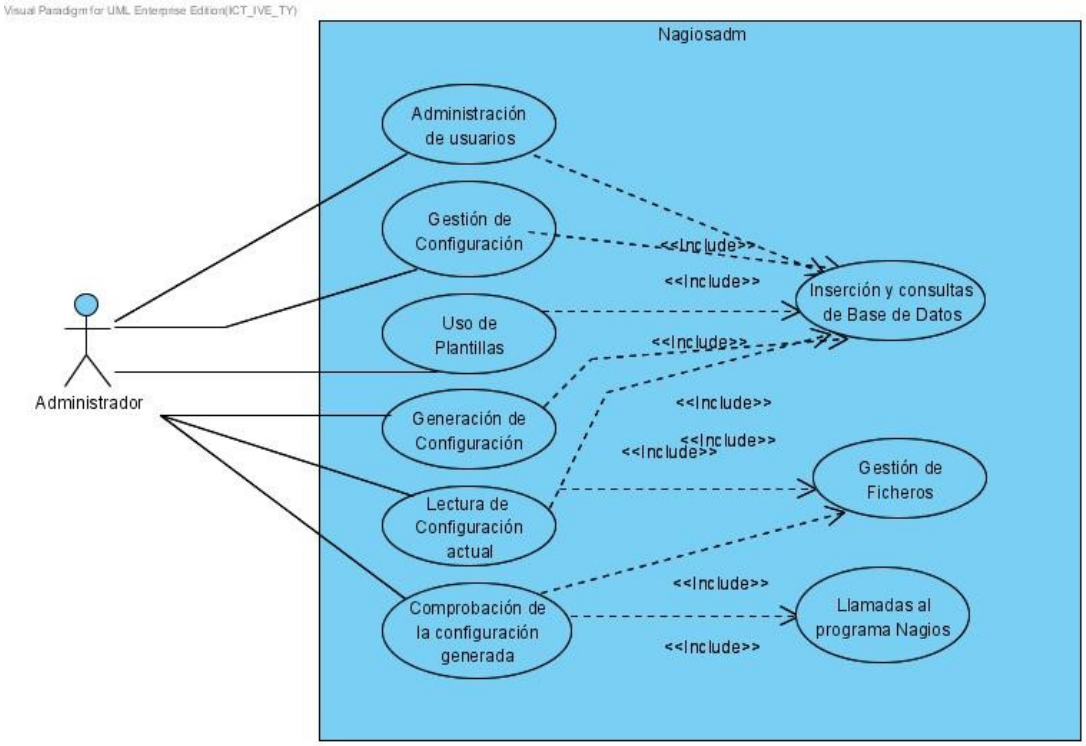


Figura 7: Diagrama de Casos de Uso

En el diagrama se pueden observar las acciones que se pueden llevar a cabo con la aplicación:

- Administración de los usuarios que pueden acceder a la aplicación para realizar cambios o visualizar la configuración.

- Gestión de la configuración de Nagios, tanto modificar los elementos

creados existentes como la creación de nuevos elementos o el borrado de otros. Estas modificaciones son comprobadas por el programa que verifica que tras ellas se sigan cumpliendo unas reglas mínimas de los atributos o que no se borre un elemento del que depende otro.

- Facilitar la configuración mediante el uso de plantillas de los objetos de configuración. Esto realmente ya es una posibilidad en la configuración de Nagios pero en un principio se pensó en que se marcase un objeto como plantilla o no y que el usuario no supiese más nada. Posteriormente se modificó la aplicación para que cuando un objeto hiciese uso de una plantilla apareciesen junto a los valores de sus atributos los valores de los atributos de la plantilla y que el administrador que configuraba el objeto eligiese si el actual objeto usaba el valor de la plantilla o lo "machacaba" con uno propio haciendo uso de un casillero.

- Comprobación de la validez de la configuración generada. Esta operación se hace sobre los archivos de configuración generados por Nagios, y no es que se verifique si la aplicación ha cometido errores sintácticos en la generación de estos archivos, sino que se invoca al programa Nagios para la verificación de los archivos para informar al usuario administrador de si la configuración actual tiene errores de coherencia y por lo tanto no podría ser usada para arrancar Nagios.

- Las operaciones citadas anteriormente lógicamente conllevan una interacción con una base de datos por parte del programa en la cual guarda toda la información relativa a la configuración y a los usuarios que pueden acceder a la aplicación.

- Salvaguardia y restauración de archivos de configuración. Se ha incluido la posibilidad de hacer copias de seguridad de los archivos de configuración generados. Estas copias son realizadas por clases de java que hacen compresión de los archivos de configuración a archivos ".zip" y posteriormente estos son movidos a una carpeta de salvaguardia. Desde la aplicación estos archivos son posteriormente listados para el usuario y se le permite a éste que seleccione uno para que la aplicación lo descomprima y obtenga los archivos de configuración.

- Comprobación de la validez de la configuración generada en los archivos, para lo cual la aplicación se apoya en el propio Nagios, el cual tiene una función de testeo de la configuración con la que se informa al administrador.

7.3 Paquetes de la aplicación

Siendo una aplicación J2EE esta va a tener una parte dedicada a la presentación Web formada por páginas jsp y html en la carpeta Web-Content y una carpeta "src" en la que están contenidos los paquetes y clases java de la aplicación.

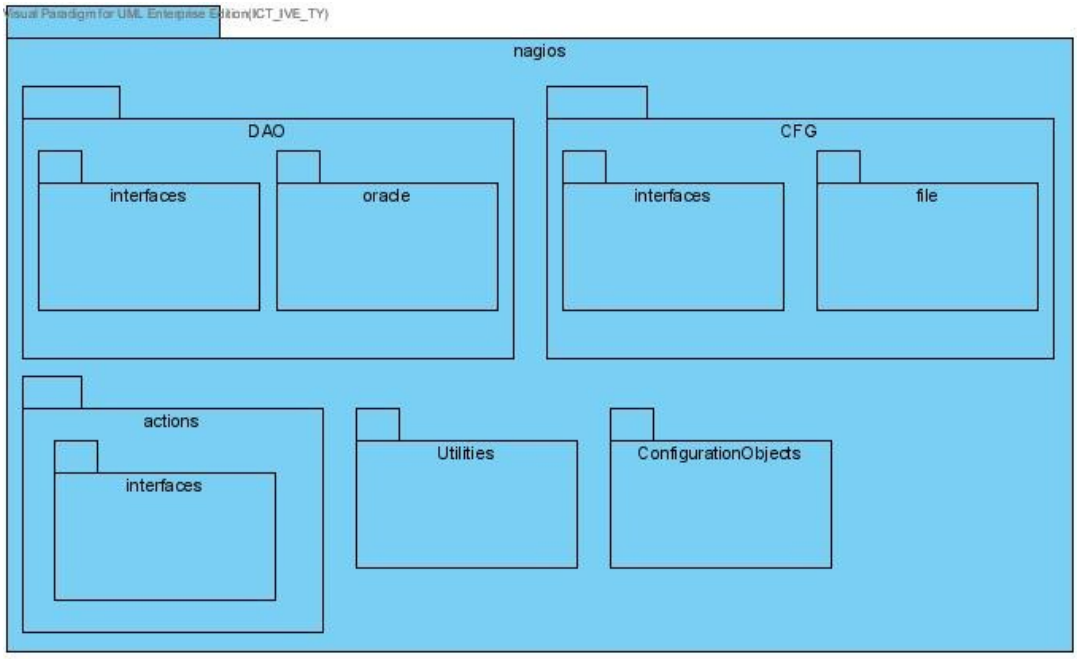


Figura 8: Diagrama UML de la estructura de paquetes de la aplicación

Las clases de la aplicación se han divido en paquetes según funciones distintas que tenían que realizar éstos. La siguiente imagen muestra la jerarquía de paquetes de la aplicación:

Para cada conjunto de clases se han usado paquetes en los que definimos las interfaces que han de cumplir estas clases y que ayudan a diseñar la función que queremos que cumplan inicialmente.

En los siguientes puntos se detalla un poco más el contenido de cada paquete y la función que realiza dentro de la aplicación.

7.3.1 Paquete nagios.ConfigurationObjects

La herramienta tiene como propósito la generación y gestión de la configuración de Nagios, por esta razón se han creado los objetos que tenemos dentro de este paquete, los cuales simulan los elementos de configuración de Nagios expuestos en el apartado 2.4.4.

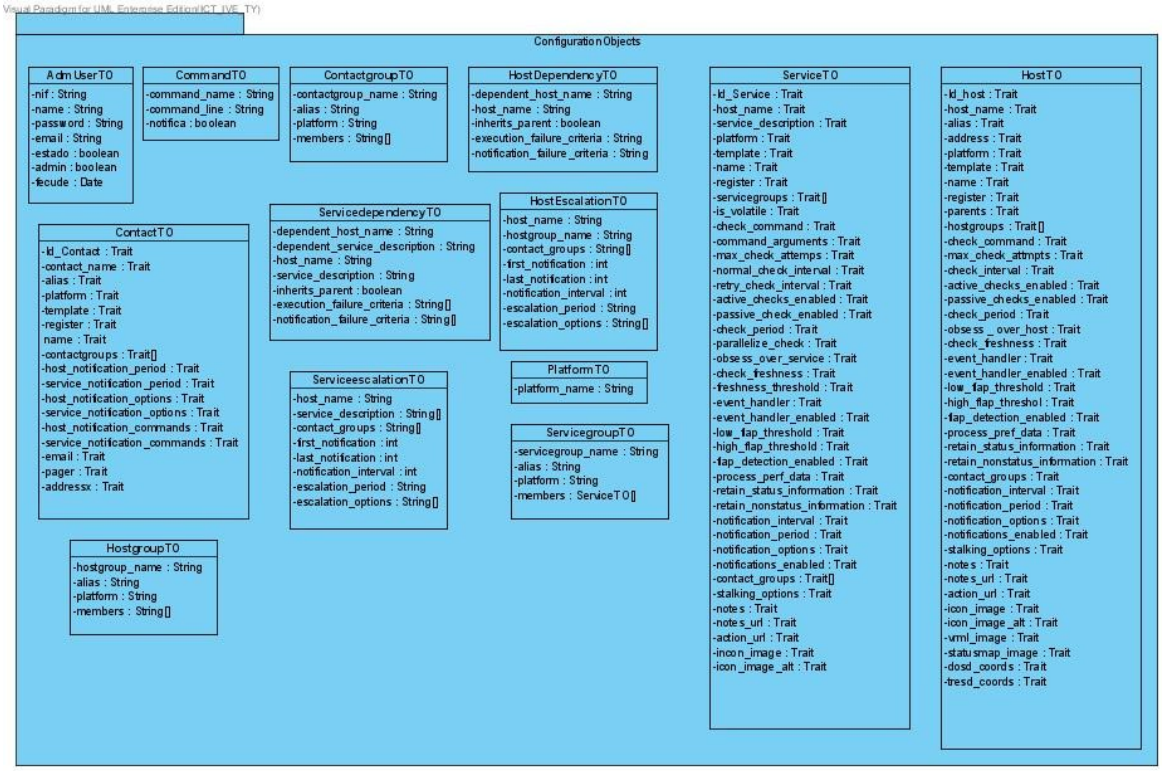


Figura 9: Diagrama UML, paquete nagios.ConfigurationObjects

Para comodidad en el análisis en el diagrama se han suprimido los métodos de las clases del tipo getTrait() y setTrait(Trait ...). Cada atributo llevaría asociados cada uno de estos métodos.

Estos objetos de configuración tienen los mismos parámetros que los que tienen los elementos de Nagios más algunos adicionales que aportan otras funcionalidades en la

aplicación. Por petición de los futuros usuarios se han adaptado estos "atributos" de cada objeto de configuración para que guarden información adicional y no sólo un valor que sería el que se escribiese en el archivo de configuración cuando se generase la configuración para que la adoptase Nagios. Para conseguir esto, en vez de usar Strings para los atributos se ha creado un tipo de objeto llamado "Trait" en otro paquete, el cual además de almacenar el valor del atributo almacena también un campo que índica si se puede sobrescribir el campo de la plantilla que importe el objeto de configuración. Además se deja abierta de una forma sencilla la posible ampliación y adición de datos relacionados con el valor del atributo, como su última fecha de modificación o el usuario que lo modificó.

En la configuración de Nagios se nos indica para cada objeto de configuración cuales de sus parámetros son obligatorios y cuales son opcionales. Esta información no esta incluida en los objetos de configuración creados en la aplicación, sino que estará en la base de datos y se harán comprobaciones en los procesos de inserción a la misma.

7.3.2 Paquete nagios.actions

Este es el paquete encargado de la ejecución de cada una de las peticiones hechas a la aplicación. Dentro de este paquete tendremos un paquete "interfaces" en el que se encuentra la interfaz de la que heredan todas las clases del paquete nagios.actions salvo ActionFactory.

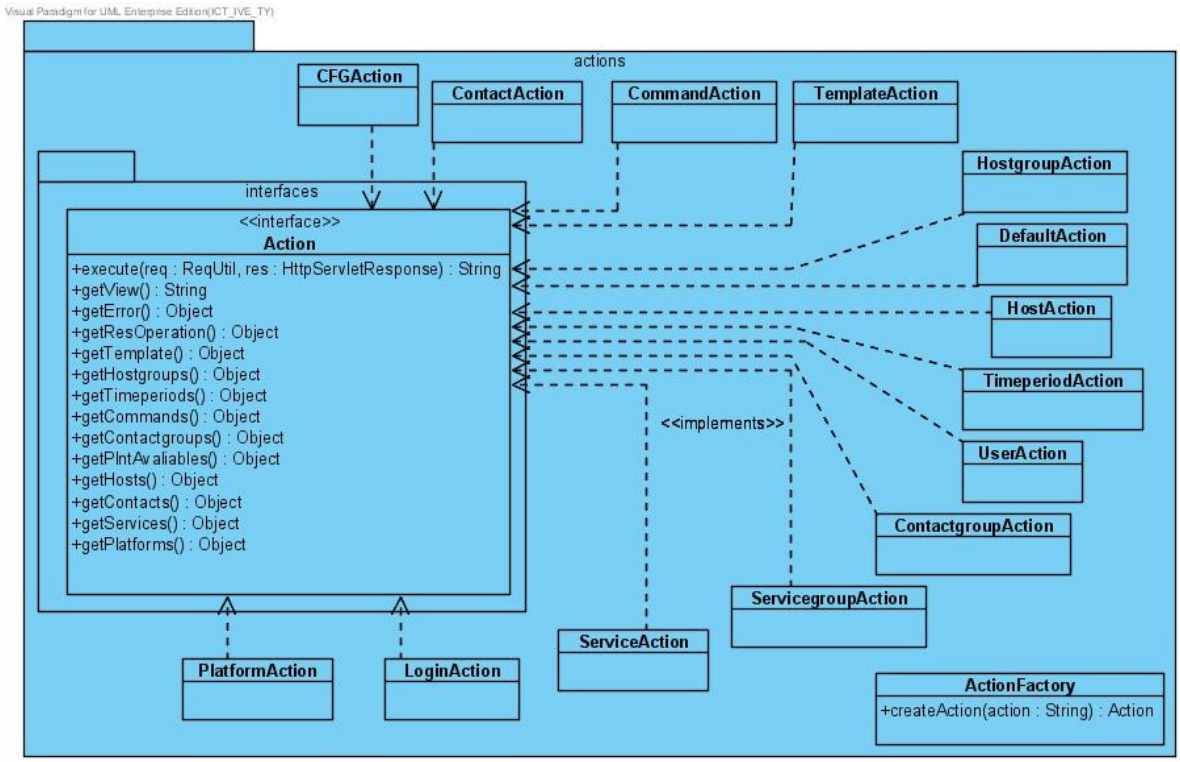


Figura 10: Diagrama UML, paquete nagios.actions

La clase ActionFactory se encarga de instanciar la acción oportuna discriminando entre las posibles mediante la cadena que se para a su método createAction(String action). El resto de clases tienen como método principal el método execute(), el cual se encarga de hacer todas las operaciones que conforman la llamada a la aplicación.

Estas llamadas a la aplicación conllevan en la mayoría de los casos la recogida de datos y presentación al usuario. Para ello cada clase tiene los atributos: error, view, resOperation, template, hostgroups, timeperiods, commands, contactgroups, plntAvaliables, hosts, contacts, services y platforms. Estos atributos no se han incluido en el diagrama UML para no sobrecargarlo y facilitar el análisis. Se puede ver que con estos atributos van relacionados los métodos "get..." que cada clase hereda de la interfaz Action. Durante la ejecución de la acción, si ha sido necesario se guarda en estos atributos los datos necesarios y después, desde la clase de control se llama a los métodos "get..." de la acción ejecutada para enviar los datos generados al usuario.

7.3.3 Paquete nagios.DAO

El paquete nagios.DAO se divide a su vez en dos paquetes, uno de ellos es el paquete "interfaces" en el que tenemos definidas las interfaces que cumplen las clases que se encuentran el paquete oracle. Además tendremos la clase DAOFactory que se encarga de instanciar la clase correcta teniendo en cuenta la base de datos que vayamos a usar. Para el caso del desarrollo tenemos el paquete oracle porque es nuestra base de datos y la clase DAPFactory estaba configurada para que llamase a las clases de este paquete.

La organización de este paquete está pensada para tener un paquete de interfaces independientes de la base de datos usada y uno o varios paquetes que implementan las clases de la interfaz para que se ejecuten teniendo en cuenta las variaciones que puedan haber en el acceso a otra base de datos.

En la siguiente figura tenemos un diagrama UML reducido del paquete comentado, no se incluyen los atributos de los métodos de las interfaces para no complicar la lectura del mismo.

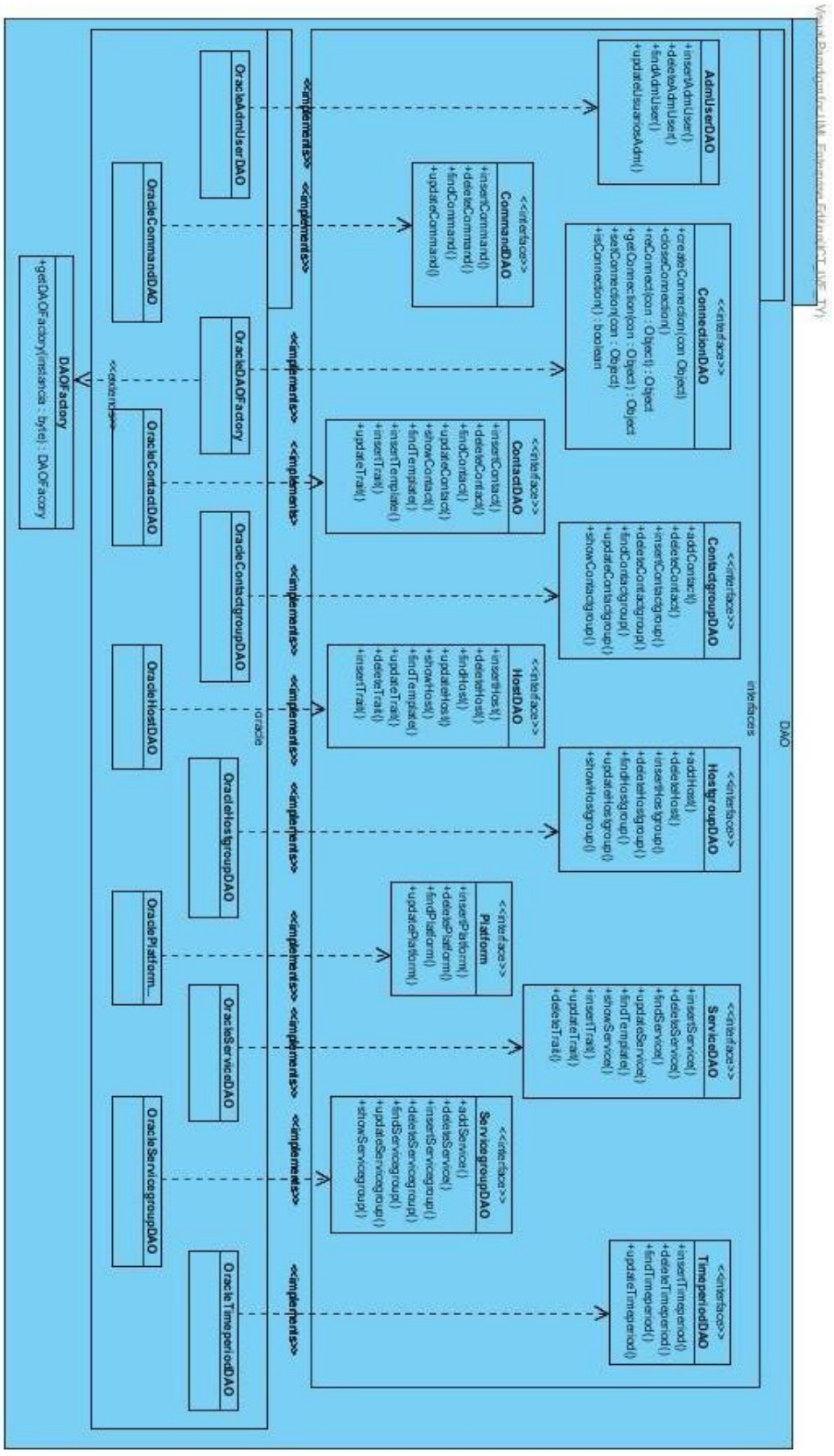


Figura 11: Diagrama UML, paquete nagios.DAO

Se puede ver que tenemos una interfaz (y una clase que la implementa) para cada objeto de configuración u objeto que se almacene en la base de datos. Cada interfaz define métodos para crear, actualizar, borrar o encontrar elementos de configuración en la base de datos. En la siguiente figura podemos ver la interfaz, con todos los atributos correspondientes, que se encarga de definir los métodos necesarios para la gestión en la base de datos del objeto de configuración Contact.

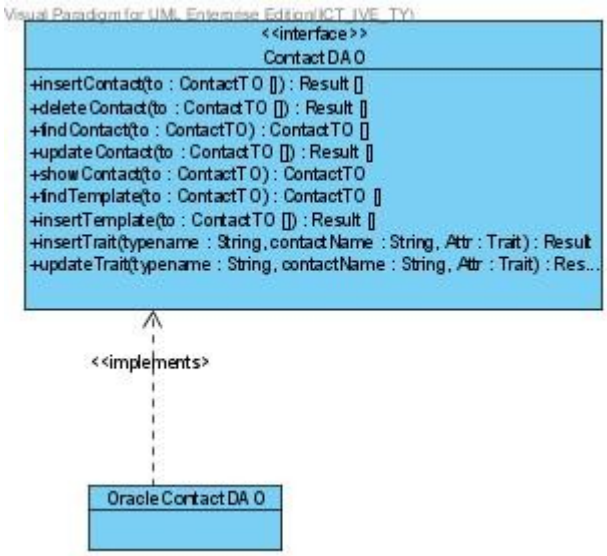


Figura 12: Diagrama UML, interfaz ContactDAO

Para el resto de las clases los atributos de entrada y de salida serán los correspondientes a los objetos que gestionen en la base de datos. Estos arrays de objetos de configuración son los objetos que se vieron en el punto 7.3.1 en el paquete nagios.configurationObjects. Los métodos que implementa cada interfaz se encargan de, según el caso, insertar todos los atributos del objeto de configuración en la base de datos, actualizar los atributos del objeto, borrarlos, etc.

7.3.4 Paquete nagios.CFG

En este paquete tenemos las interfaces y clases relativas al manejo de los archivos en los que se escribirá la configuración o de los que se leerá. Dentro de este paquete tenemos otros dos paquetes, uno es el de interfaces, en el que se define para cada objeto de configuración una clase con los métodos que habría que implementar para la lectura y escritura de ese objeto de configuración en un archivo, y el otro paquete contiene las clases que implementan las clases definidas en el paquete de interfaces. Adicionalmente tenemos una clase independiente llamada CFGFactory que es la encargada de la apertura, cierre, lectura y escritura de los archivos de configuración. Las clases que escriben la configuración se sirven de esta clase común para abrir cada una su archivo correspondiente y escribirlo o leerlo.

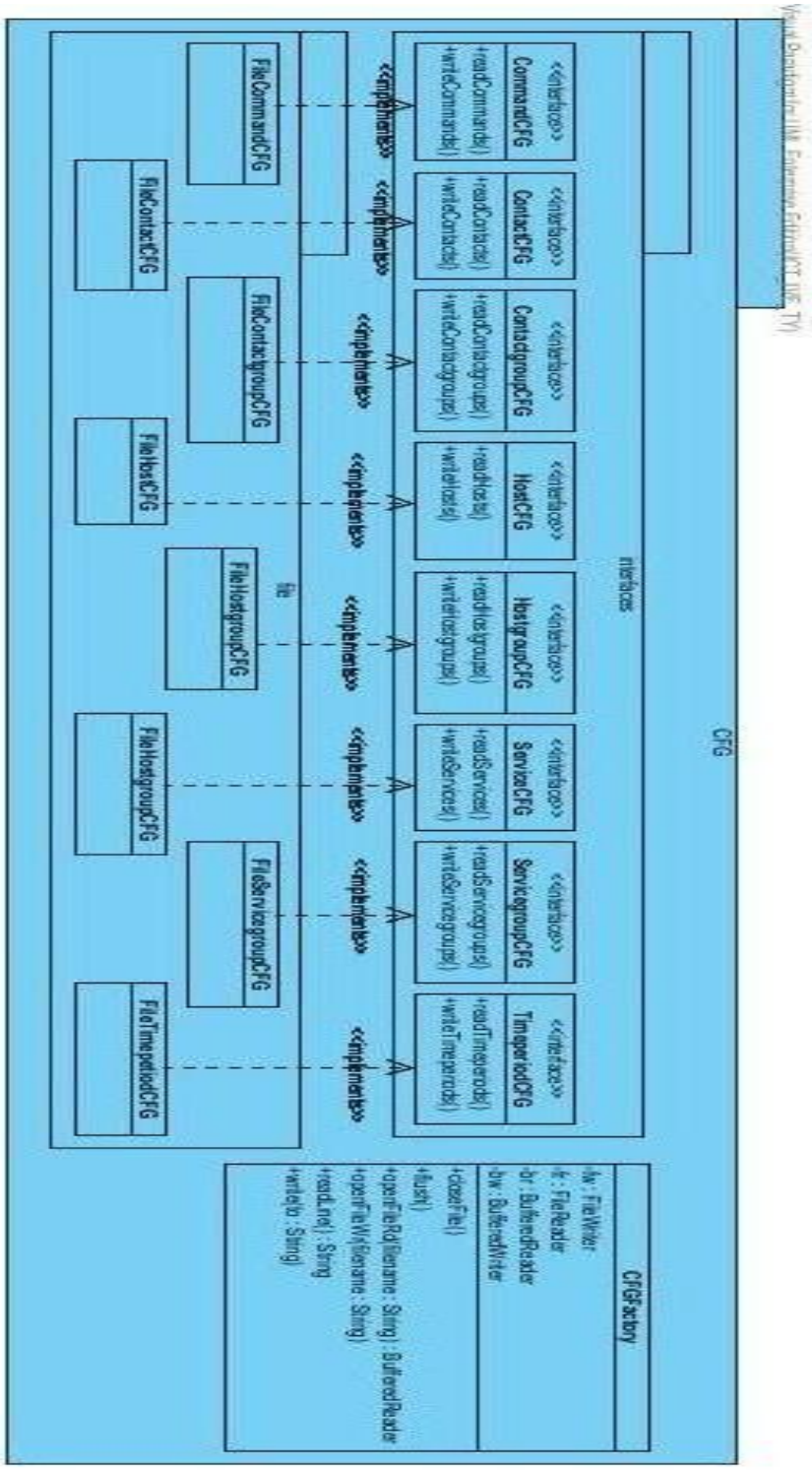


Figura 13: Diagrama UML, paquete nagios.CFG

Nuevamente en el diagrama UML no se han representado toda la información relativa a los métodos que definen la interfaz. Si ampliamos para la interfaz que define la escritura y lectura de los objetos de configuración Command tenemos lo siguiente:

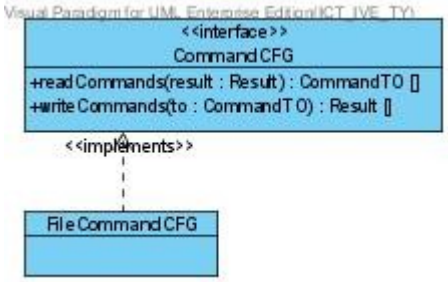


Figura 14: Diagrama UML, interfaz CommandCFG

Para cada uno de los otros objetos tendremos una interfaz que define un método que escribe al que se le pasa un array con los objetos que se quieren escribir y un objeto que lee que devuelve un array con todos los objetos de configuración que ha leído de un archivo de configuración.

La implementación de cada uno de estos métodos sigue fielmente la documentación de Nagios y va escribiendo objeto a objeto con las etiquetas de cada atributo y el valor que tiene ese atributo para el objeto en la base de datos. En la lectura se tomando de cada linea la etiqueta del atributo y el valor y haciendo comparaciones para ver que atributo se está leyendo. Se ha tenido en cuenta qué atributos son obligatorios en cada elemento de configuración y se hace una comparación al final de la lectura de cada elemento para verificar que el archivo de configuración que se está pasando a la aplicación es correcto. También se han tenido en cuenta problemas que pudiesen suponer espacios en blanco y lineas con comentarios.

7.3.5 Paquete nagios.Utilities

En el paquete de utilidades se han agrupado las clases comunes de la aplicación y que no tenían unas funciones concretas como para agruparlas dentro de otro paquete ni una

elevada importancia como para crear un paquete específico para cada una. No obstante en este paquete tenemos clases que son fundamentales en la aplicación y que desempeñan labores imprescindibles en el transcurso de cada ejecución de una acción en la aplicación.

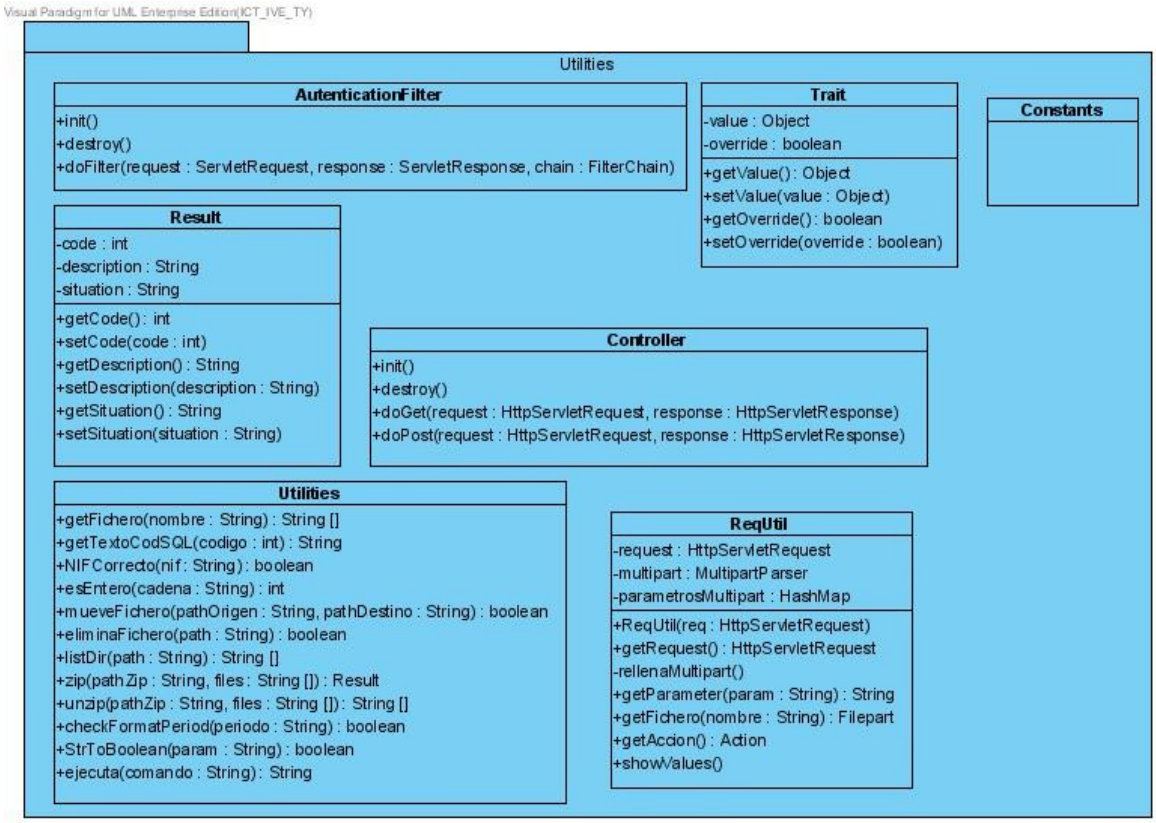


Figura 15: Diagrama UML, paquete nagios.Utilities

La clase de utilidades es una clase que no es propia de la aplicación, se ha tomado de otra. En la clase se tenían métodos que eran necesarios y se han añadido otros para la gestión de los ficheros o pasar de tipo String a boolean.

La clase ReqUtil se utiliza para una gestión mejorada de los atributos que se pasan a la aplicación desde un formulario.

La clase AutenticationFilter es una clase que está configurada en el archivo "web.xml" (archivo de configuración de la aplicación) para que actúe como filtro de la aplicación, esto es, esta clase siempre será ejecutada en cada llamada a la aplicación. Por lo tanto usamos esta clase como filtro de seguridad para la aplicación y en ella verificamos

que el usuario se ha autenticado anteriormente, si no se ha autenticado se redirige a la acción de Login.

La clase Controller es, como su nombre indica, la clase que controla todas las llamadas a la aplicación. Todas las llamadas a la aplicación son hechas al controlador y le pasan un parámetro "action" que es el que usa la clase Controller para lanzar la ejecución apropiada de la aplicación. Esta clase lee el parámetro de acción, instancia la acción, llama a su método execute, recoge todos los objetos que se hayan generado tras haberse ejecutado la acción y después de esto redirecciona la llamada a la página jsp correspondiente. En la redirección se pasan en el request todos los objetos de datos que se hayan generado en la ejecución de la llamada y es el código java de la página jsp el que se encarga de mostrar los datos.

Los objetos Trait y Result han sido generados para dar más funcionalidad a la gestión de los atributos de los objetos de configuración en el caso de Trait, y para dar más información acerca de la ejecución de ciertas acciones en el caso de Result. Como se comentó antes, en Trait añadíamos más atributos además del valor del atributo y además dejamos abierta la posibilidad de seguir añadiendo ya que usamos un objeto y no un String para los valores en la ejecución de la aplicación. En el objeto Result tenemos además de un código que nos informa del error que se ha producido, una descripción y una situación que en ciertas ejecuciones se rellenará indicando al usuario en que parte ha tenido un error la aplicación o para indicar donde existe un error en un archivo de configuración que está leyendo la aplicación. La clase Result tendrá grabados los códigos de error SQL y relacionados con el significado que ha de devolver en el log de la aplicación.

7.4 Ejecución de una llamada a la aplicación

Es interesante en la aplicación la forma de tratar una llamada a la misma. En el siguiente diagrama UML podemos ver la lógica que sigue dicha llamada:

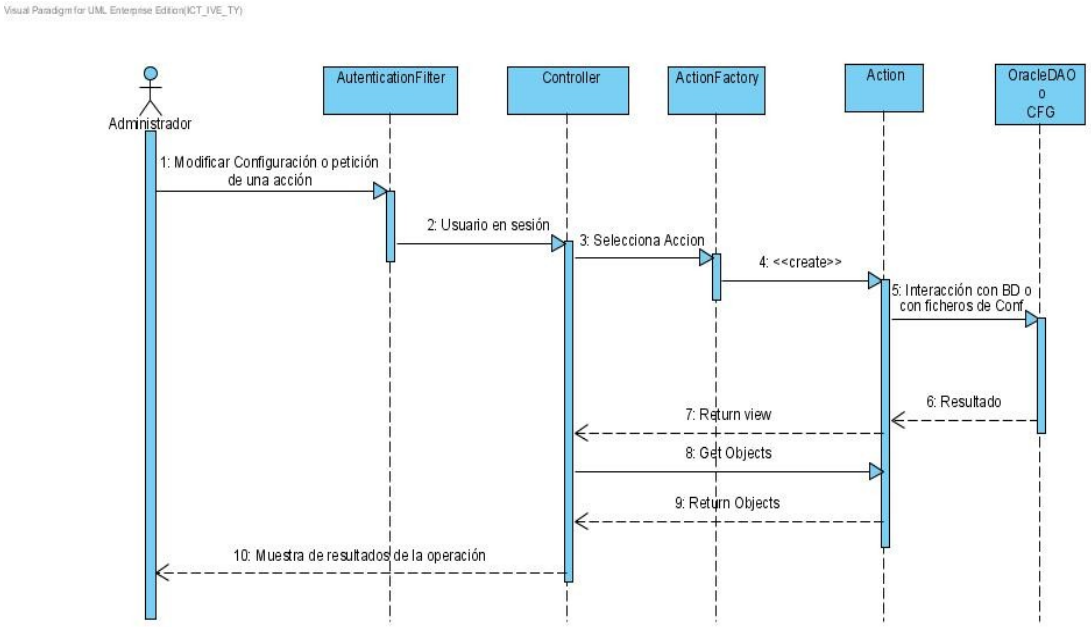


Figura 16: Diagrama UML, ejecución de una llamada

El diagrama de secuencia anterior es un diagrama genérico que nos introduce en la secuencia que sigue una petición cuando es recibida por la aplicación. El usuario que genera la petición es un usuario administrador que está haciendo modificaciones en la configuración de Nagios, la está cargando en la aplicación o la está generando, la está generando a partir de la información contenida en la base de datos, etc. Detallamos a continuación los siguientes pasos internos que ocurren en la aplicación:

- Ejecución de la clase AutenticationFilter. Debido a la configuración de la aplicación en el archivo WEB-INF/web.xml tenemos esta clase configurada como filtro de la aplicación, y como tal se ejecutará primeramente antes que cualquier página o servlet que sea llamado. La clase AutenticationFilter se encarga de verificar que el usuario tiene una sesión creada con la aplicación y si no es así cambia la petición que haya iniciado el usuario por una petición a la acción de Login. Así tenemos un muro de seguridad en el acceso a la aplicación.

- La clase ActionFactory se encarga de instanciar una acción a partir del parámetro "action" que ha sido pasado a la clase Controller. Simplemente se hacen comparaciones de cadenas para saber qué clase Action hay que instanciar.

- Ejecución de la acción. A continuación de instanciar una clase Action se llama a su método "execute", el cual es la verdadera ejecución de la llamada a la aplicación. En este punto es donde damos los pasos necesarios para realizar la petición del cliente, como podrían ser actualizar ciertos parámetros de un objeto o lanzar una petición de borrado de otro. Para los accesos a la base de datos o a los ficheros se usan otras clases, no se hace en la clase directamente.

- Clases de Acceso a la base de datos y a ficheros. En estas clases tenemos las operaciones básicas de lectura o escritura de archivos y para el caso de la base de datos tendremos las instrucciones SQL para consultas, borrado, inserciones o actualizaciones de las tablas de la base de datos.

- Ejecución de la clase Controller. Todas las llamadas de la aplicación están hechas a la clase Controller pasándole un parámetro "action". Las llamadas no son hechas a las páginas jsp porque éstas por si solas no desencadenan ninguna ejecución importante en la base de datos. La clase Controller es la encargada de instanciar un ActionFactory y de llamar al método "execute" de la acción que se haya creado. Después de que la acción se haya ejecutado y haya devuelto una vista que haya que mostrar al usuario o null si no hay que mostrarle una página en concreto, la clase Controller se encarga de preguntar por todos los objetos posibles a la instancia de la acción para adjuntarlos a la página necesaria o para comprobar si hay error. Si ha habido error u objeto de resultados se redirecciona al usuario a la página de error o a la de resultado genérico que son las encargadas de mostrarlos. Si la cadena vista tiene contenido, se redirecciona al que se indique y se incluyen los objetos que se hayan podido crear en el response de la llamada para que la página jsp a la que se redirecciona los pueda mostrar.

7.5 Acceso a la Base de Datos

Para cada uno de los objetos de configuración tenemos una interfaz en el paquete nagios.DAO.interfaces (DAO son las iniciales de Data Accesos Object) en el cual se definen los métodos necesarios para el acceso a la base de datos gestionando ese objeto de configuración.

Normalmente estos métodos serán de inserción, borrado, búsqueda y muestra completa del objeto, aunque en ciertos casos se añadirán algunos más necesarios para añadir funcionalidades como la búsqueda de plantillas de ese objeto o la gestión individual de atributos del objeto. Esto ocurrirá en objetos de configuración complejos como los Services, Hosts y Contacts. Estos métodos definidos para la gestión de objetos de configuración en la base de datos están dispuestos para usar "arrays" de estos objetos en la inserción, borrado o búsqueda.Esto es así, además de porque cuando se busque se podrán encontrar muchos objetos que se adapten a determinada búsqueda, porque cuando se inserten objetos en la lectura de archivos o se extraigan habrá que hacerlo con muchos elementos de cada objeto de configuración y usando arrays no tendremos que estar llamando a un método dentro de un bucle.

En el paquete nagios.DAO.oracle están las clases que implementan las interfaces anteriormente comentadas para la gestión de los objetos de configuración en una base de datos Oracle. Como se comentó en el apartado en el que hablábamos sobre el paquete nagios.DAO se disponen las interfaces en un paquete y se espera que haya otros paquetes que implementen las interfaces para las distintas bases de datos a las que se acceda si es que hay diferencias en el acceso.

7.5.1 Modificaciones en el acceso para MySQL

Acabada la implementación de la aplicación se intentó desplegar usando otra base de datos distinta de Oracle, se usó una base de datos MySQL. Al hacerlo la aplicación comenzó a dar varios errores y se tuvieron que hacer algunos cambios en las instrucciones de lenguaje sql que java invocaba sobre la base de datos.

Se tuvo que modificar la clase OracleDAOFactory para realizar la conexión de manera distinta. Los aspectos a cambiar en las clases de acceso a la base de datos son sólo de semántica en las sentencias SQL. Detallamos a continuación los más relevantes.

Se añade un campo password al usuario administrador y se modifica fecude para que sea un string, por lo que hay que cambiar la inserción de usuarios y la extracción de sus datos en la clase que accede a objetos AdmUser. Da problema si se deja fecude como date y no se rellena, por eso se cambia a String.

Hay que modificar las comparaciones con NULL, ya que MySQL las interpreta de forma distinta a como lo hace Oracle. Las sentencias que incluyen WHERE ... IS NOT NULL habrá que cambiarlas por WHERE ... != “” . En Oracle que el campo esté vacío se interpreta como null, pero en MySQL se interpreta como vacío y se usa un valor para null.

Además habrá que cambiar las instrucciones de extracción de un número en la secuencia de Oracle por las instrucciones para acceder a una tabla en MySQL que emula una secuencia.

7.6 Escritura y Lectura de Archivos de Configuración

Para estas acciones se ha usado la misma arquitectura de clases que en el acceso a la base de datos, es decir, se ha creado un paquete en el que tenemos las interfaces de acceso a la configuración para cada objeto de confguración, nagios.CFG.interfaces, y éstas son implementadas por clases en el paquete nagios.CFG.file.

En estas interfaces se definen dos métodos, uno de lectura de la configuración y otro de escritura. Adicionalmente a las interfaces tendremos en el paquete nagios.CFG la clase CFGFactory que gestiona el acceso a los archivos tanto en modo lectura como escritura. Los métodos de escritura y de lectura se apoyarán en esta clase para el acceso a los archivos y además usarán el método “toString” de cada objeto para poder plasmar la configuración almacenada en los archivos de configuración. En el caso de la lectura la clase que se encarga de la misma deshecha los comentarios y las lineas en blanco, de la

parte útil se obtiene cada elemento configurado en el archivo y finalmente usando las clases de acceso a la base de datos se guardan todos los objetos de configuración leídos.

PARTE IV : CONCLUSIONES Y LINEAS FUTURAS

8 Conclusiones

El objetivo de este proyecto era tener una aplicación que gestionase la aplicación Nagios desde una interfaz gráfica que ofreciese un mejor entendimiento para el administrador. Este objetivo se ha cumplido usando una aplicación J2EE y una base de datos Oracle, dos de los requisitos principales que se debían cumplir.

El proyecto ha supuesto un reto en cuanto a comenzar a programar una aplicación J2EE con unos conocimientos del lenguaje java básicos. Aunque se ha tomado otra aplicación como base para algunas funciones y se ha podido reutilizar código, ha sido necesario un fuerte estudio de java y la estructuración de una aplicación J2EE.

Además de cumplir el objetivo principal de tener esta interfaz gráfica de configuración se han experimentado con otras funciones de java y se han añadido utilidades adicionales a la aplicación. La ejecución del proyecto, además de ser exitosa ha sido de gran ayuda para conseguir una formación en el ámbito de creación de este tipo de aplicaciones.

9 Futuras Ampliaciones

Durante la implementación práctica del proyecto han ido surgiendo multitud de ideas que serían aplicables a la aplicación para permitirle más funcionalidades, mejorar su seguridad o hacerla más eficiente en cuanto a reducción de código o reducción de procesado. Algunas de ellas se aplicaron durante la realización, como fue el caso de la generación de copias de seguridad de la configuración generada por la aplicación, otras las detallamos a continuación:

- Visualización de la configuración relativa a los objetos para los que un usuario de administración está configurado como contacto. Esto es, la configuración podría partir del email del usuario administrador que está accediendo a la configuración y mostrarle sólo la configuración de los objetos que le enviarían notificaciones para que pudiese modificar sólo éstos.

- Creación de usuarios con distintos niveles de administración. Además de un usuario administrador como el que accede a la aplicación actualmente que puede cambiar todos los atributos de los objetos, crear nuevos objetos o borrarlos, se podrían configurar otros usuarios que sólo pudiesen cambiar parámetros menos relevantes como son los tiempos entre envíos de notificaciones, los periodos de chequeo u otras configuraciones que no fuesen tan vitales, además de no permitirles el borrado ni la creación de objetos.

- Parada de Nagios, sustitución de la configuración y arranque con la configuración generada. Actualmente sólo se genera la configuración y se ha preferido tenerla a parte, que no sustituya la de Nagios y que sea el administrador el que haga esto y reinicie el servicio.

- Modificación de la interfaz gráfica para permitir que las etiquetas de cada atributo durante la generación y actualización de objetos sean links a las páginas de ayuda de Nagios y concretamente al punto en el que se explica dicho objeto. En la aplicación actual se ha incluido la ayuda y un enlace a la misma para que el administrador la pueda ir visualizando y le sirva de apoyo, aunque con esta modificación se realizaría una búsqueda más directa de la información.

10 Bibliografía

- Documentación de Nagios 2.x [en linea]. <http://nagios.sourceforge.net/docs/2_0/>

- Wikipedia – Base de datos [en linea]. <http://es.wikipedia.org/wiki/Bases_de_datos>

- Monografías.com – Bases de datos [en linea]. <http://www.monografias.com/trabajos11/basda/basda.shtml>

- Fábrica Nacional de moneda y timbre [en linea]. <http://www.fnmt.es/es/html/id/sc_idid03.asp>

- CERES [en linea]. <http://www.cert.fnmt.es/index.php?cha=gen>

- El Pais. Artículo: Una identidad en la red. [en linea].
<http://www.elpais.com/articulo/internet/identidad/red/elpfot/20060425elpepunet_4/Tes>

- SUN: The J2EE 1.4 Tutorial [en linea].
<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/>

- Arlow, Jim. UML2. Madrid : Anaya Multimedia, 2006. ISBN 84-415-2033-X

PARTE V : GUIA DE INSTALACIÓN

11 Instalación

En este punto se explican los pasos básicos para que un usuario pueda instalar la aplicación, configurarla y ejecutarla en una máquina para manejar la configuración de Nagios.

Los pasos e instrucciones que se detallan a continuación dependerán en algunos casos del sistema operativo sobre el que se está desplegando la aplicación. En la documentación se expondrán los pasos suponiendo que se está usando un sistema UNIX, como una distribución de Linux, que es lo que se ha usado para el desarrollo de la aplicación. Para los casos en los que se ha usado un software propietario se ha instalado posteriormente alguno similar de fuentes abiertas.

11.1 Base de datos

Se da por supuesto que se tiene instalada una base de datos. Para la base de datos se usará un script que genere la estructura de tablas que usa la aplicación, este script se generó para la base de datos que se utilizó en desarrollo, la cual era una base de datos de Oracle, posteriormente se ha adaptado y el que se tiene adjunto con el código de la aplicación se puede usar con una base de datos de MySQL.

En los siguientes pasos se han usado una serie de nombres para la base de datos y los usuarios. Estos no tienen por qué conservarse, aunque si se van a modificar se han de cambiar también en el código de la aplicación, en la clase que realiza la conexión a la base datos, y cambiar el nombre de la misma, usuario y contraseña para que la conexión se realice correctamente. Esta clase en la que se realiza la conexión es OracleDAOFactory.java en el paquete nagios.DAO.oracle, pero podemos configurar los parámetros de conexión en el archivo de configuración “web.xml”, en este configuraremos los parámetros de conexión:

```
<init-param>
   <param-name>mysqlDriver</param-name>
   <param-value>com.mysql.jdbc.Driver</param-value>
```

```
</init-param>

<init-param>
   <param-name>mysqlUser</param-name>
   <param-value>na_user</param-value>
</init-param>

<init-param>
   <param-name>mysqlPassword</param-name>
   <param-value>na_user</param-value>
</init-param>

<init-param>
   <param-name>mysqlURL</param-name>
   <param-value>jdbc:mysql://localhost/nagiosadm</param-value>
</init-param>
```

Para usarlo con MySQL primeramente, además de tenerla instalada, es necesario definir en la base de datos los usuarios que tienen permiso para acceder. Es recomendable instalar y arrancar un navegador y phpMyAdmin, la cual es una aplicación que nos ofrece una interfaz gráfica para configurar MySQL sin usar linea de comandos.

El nombre de la base de datos que usaremos será "nagiosadm" y crearemos dos usuarios para acceder a la misma. Primero crearemos uno con todos los permisos posibles sobre la base de datos, el cual será el propietario y lo llamaremos na_owner, el otro será el usuario que usa la aplicación, "na_user", y que sólo tendrá permisos para modificar y consultar las tablas. Le pondremos la contraseña "na_user" al usuario "na_user" para conectarse a la base de datos.

A continuación creamos las tablas necesarias para la aplicación usando el usuario na_owner. Para ello desde la linea de comandos usamos el fichero installdb.sql que se adjunta con la aplicación y lo ejecutamos con mysql haciendo que sea un fichero de entrada para que se ejecuten las lineas que contiene. con el comando:

mysql -u na_owner nagiosadm -p < installdb.sql

Tras la inserción del comando tendremos que escribir la contraseña del usuario na_owner y se ejecutarán todas las sentencias que crean las tablas. Podremos verlas creadas con phpMyAdmin. Para poder acceder a la aplicación con este script se crea un

usuario inicial en la tabla de administradores con DNI: 00000000X y contraseña “admin”. Con este usuario podemos acceder en principio a la aplicación y en la sección de usuarios crear uno nuevo y desactivar con el que hemos accedido para no tener un agujero de seguridad.

Creadas las tablas hemos de tener en cuenta ciertos aspectos que nos pueden producir incompatibilidades en el acceso a las tablas. Cuando se desarrolló la aplicación se hizo sobre una base de datos en la que no existía diferenciación entre mayúsculas y minúsculas, y por lo tanto en la base de datos que se vaya a usar para nuestro propósito tendremos que configurarla para que funcione de igual forma. En el caso de MySQL es cambiar uno de los parámetros de arranque de la base de datos. Podemos mirar la documentación de MySQL y ver que el parámetro a modificar es “lower_case_table_names”, pero ha de ser modificado en la instrucción de arranque de MySQL. Adjuntos con la aplicación tenemos los archivos de configuración del arranque de MySQL (my.cfg) y el script de arranque que se debería encontrar en /etc/init.d (mysqld).

Una vez hecho esto ya tenemos la base de datos lista y sólo nos tenemos que asegurar que esta corriendo y no tenemos problemas para acceder a la misma.

11.2 Despliegue de la aplicación

La aplicación “nagiosadm” es una aplicación J2EE, y como tal necesitará un servidor de aplicaciones en el que ser desplegada. La realización de la aplicación es independiente del servidor de aplicación y no importará el que usemos. Para la fase de desarrollo se usó el OC4J de Oracle, pero también se ha probado en Apache Tomcat 5.5.

Para el caso de Apache Tomcat bastará con pegar el archivo “nagiosadm.war” en el directorio $CATALINA_HOME/webapps, siendo $CATALINA_HOME el directorio en el que tenemos instalado Tomcat. Automáticamente el servidor desplegará la aplicación y la tendrá lista. Suponiendo que accedemos al servidor de aplicacíon por el puerto 8080

(el que tiene configurado por defecto cuando se instala), la URL de acceso a la aplicación será "http://localhost:8080/nagiosadm/"

Es importante que tengamos las librerías comunes correctas en el directorio de librerías de Tomcat. Si hubiese algún problema quedará registrado en el log de nagiosadm en la carpeta logs dentro del directorio de instalación de Tomcat.

11.3 Configuración de los directorios

La aplicación se ejecuta sobre el Servidor de aplicaciones en el que se haya desplegado, en el caso de instalación explicado es Tomcat, por lo que el usuario que genera o lee los archivos de configuración es el usuario que lanza el Servidor de aplicaciones. En una instalación habitual este usuario es el usuario tomcat u otro usuario que se haya designado para esta labor.

Si el sistema opera de una forma lógica, el usuario que lanza el Servidor de Aplicaciones, y que por lo tanto genera y lee los archivos de configuración, no ha de tener permisos sobre los archivos de configuración originales del servicio Nagios. Por lo tanto, la forma de actuar del administrador que genera la configuración de Nagios usando esta aplicación es crear unos directorios de instalación en los que se tiene la configuración para leerla, escribirla y salvarla y después sustituirla en el Servicio Nagios y reiniciar este para que lea la nueva configuración.

Cuando despleguemos el programa estará configurado para usar como directorio principal el directorio /usr/local/nagiosadm, el cual ha de configurarse con el usuario tomcat como propietario. En definitiva, los directorios que tiene configurados la aplicación nagiosadm son los siguientes:

- **Directorio de generación y lectura de la configuración**. Es importante que en este directorio existan lo archivos de configuración de nagios nagios.cfg y resources.cfg, los cuales se pueden copiar de una instalación de Nagios. El archivo nagios.cfg estará configurado para que importe todos los archivos de configuración

del mismo directorio y para que use el archivo de fuentes resources.cfg del mismo directorio. Por defecto este directorio viene configurado en la aplicación como /usr/local/nagiosadm/etc.

- **Directorio de ejecución.** En el se guarda el ejecutable de Nagios que después se utilizará para comprobar que la configuración generada es correcta. El ejecutable de Nagios se puede copiar de una instalación de Nagios y habrá que darle permisos de ejecución para el usuario del Servidor de aplicación. Por defecto este directorio viene configurado como /usr/local/nagiosadm/bin.

- **Directorio para la salvaguardias.** En el cual se crearán los archivos comprimidos .zip con la configuración. Por defecto este directorio viene configurado como /usr/local/nagiosadm/etc/save.

Se podrán crear estos directorios y darles permisos para el usuario que lanza el Servidor de aplicación o se podrá cambiar la configuración y cambiar las rutas de estos directorios. Si se quieren cambiar las rutas de estos directorios y los enlaces a los ficheros de configuración se puede hacer en el archivo de configuración web.xml de la aplicación. Junto con el código de la aplicación se incluye un archivo .zip en el que están incluidos los directorios antes mencionados (los configurados por defecto) conteniendo una configuración inicial de Nagios como ejemplo y algunas salvaguardias hechas. Para instalar los directorios y dejarlos en una situación idónea habría que situarse en nivel más superior, descomprimir el archivo nagiosadm_dir.zip y darle a la estructura generada permisos para el usuario tomcat:

cd /

unzip nagios_dir.zip

chown -R tomcat@tomcat /usr/local/nagiosadm

11.4 Usuarios de administración

Como último paso para la configuración de la aplicación se han de configurar los usuarios que acceden a la aplicación para la gestión de la configuración de Nagios. Se ha configurado el script de la base de datos para que añada un primer usuario de acceso a la aplicación, para así poder acceder a ella por primera vez. Este usuario es el usuario "nagiosadm" con contraseña "admin".

Una vez hemos accedido podemos cambiar la configuración de los usuarios que acceden a la aplicación en la sección de "Gestión de Usuarios". En ella podemos dar de alta los usuarios que vayan a acceder a la aplicación y desactivar el usuario "nagiosadm" para que no tengamos un agujero de seguridad en la aplicación.

PARTE VI : MANUAL DE USUARIO

12 Manual de Usuario

La aplicación Nagiosadm tiene un manejo sencillo, la interfaz de entrada está dividida en varias secciones que permiten al usuario configurar Nagios y gestionar los archivos de configuración sin tener que acceder prácticamente a la consola o a los archivos generados.

12.1 Uso de la aplicación

Una vez autenticados en la aplicación accederemos a ésta y nos ofrecerá una interfaz gráfica sencilla con un menú lateral que nos servirá para acceder a las distintas secciones. En la siguiente figura mostramos la pantalla de bienvenida de la aplicación.

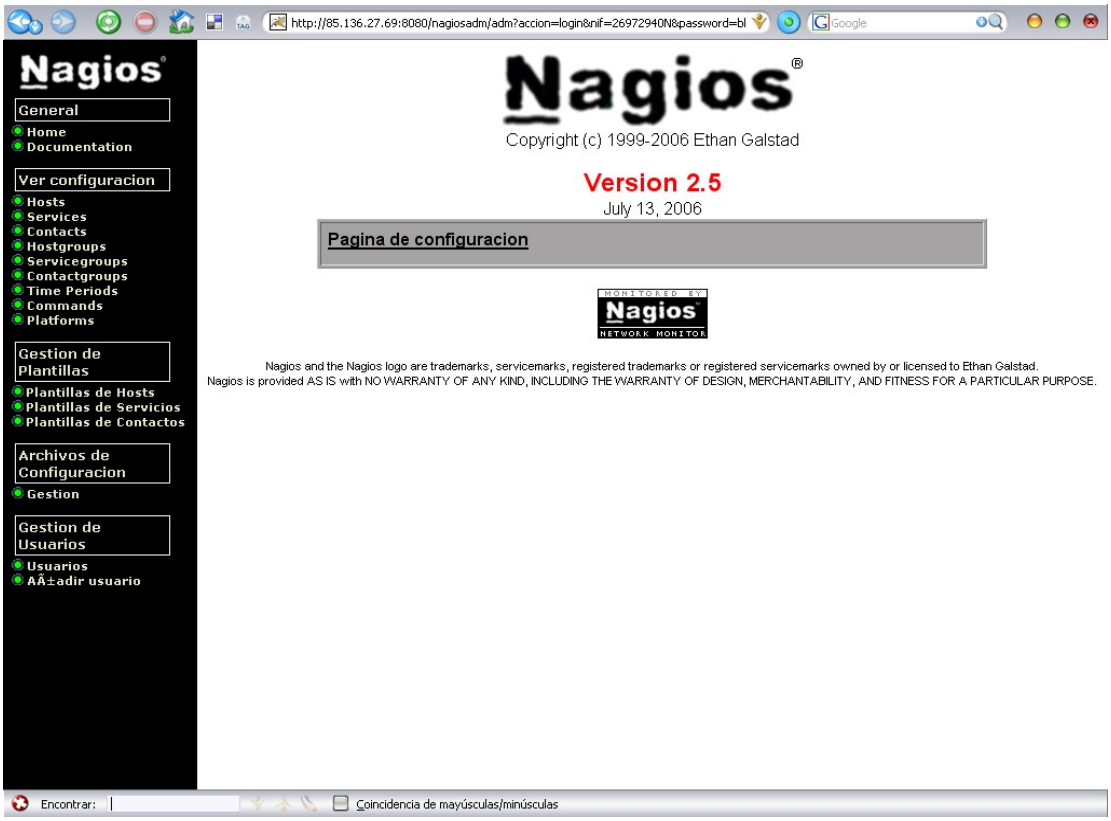


Figura 17: Interfaz de usuario de Nagiosadm

Podemos ver en el frame lateral izquierdo un menú dividido en las siguiente partes:

- **Ver Configuración**: En esta sección tenemos los distintos enlaces para cambiar y crear los objetos de configuración de Nagios. Al pinchar en los enlaces que se muestran en esa sección se listarán los objetos, se permitirá añadir nuevos objetos y borrar alguno de los añadidos.

- **Gestión de Plantillas**: Aquí tenemos una gestión a parte de las plantillas que queramos usar como objetos de configuración. Como permite la configuración de Nagios, en Nagiosadm también podremos usar objetos de configuración como plantillas. Para ello tendremos que configurarlo en los objetos. En esta sección veremos las plantillas ya configuradas y podremos crear algunas nuevas.

- **Archivos de Configuración**: En esta sección el único enlace que tenemos se llama gestión. En la página de gestión de la configuración podremos ordenar que la configuración se escriba en los ficheros, se cargue de los ficheros, se compruebe la configuración escrita en los ficheros, se salve en archivos .zip o se cargue de los mismos. Es aconsejable que antes de escribir la nueva configuración que tengamos guardada en la base de datos salvemos la que tenemos en los archivos, así tendremos un histórico y podremos volver a versiones anteriores de la configuración si observamos errores.

- **Gestión de Usuarios**: En esta sección podremos ver los actuales usuarios de administración que pueden acceder a la aplicación Nagiosadm y podremos añadir nuevos usuarios. Para los nuevos usuarios lo más importante es el DNI del usuario y la clave del mismo, por lo que se tiene una función que comprueba que el DNI es correcto y también se comprueba que la clave tenga un mínimo de seis caracteres. En los usuarios que tenemos añadidos tenemos la posibilidad de desactivarlos para que aunque estén añadidos se permita su acceso a la aplicación o no.

Los objetos que pueden usar plantillas son los Contacts, Hosts y Services. En las páginas en las que configuramos o añadimos estos objetos podemos usar las plantillas que

hayamos creado. Para ello se ha dispuesto una columna paralela a la de los valores del propio objeto en la que podemos ver los valores de la plantilla que usa. Para cada atributo del objeto tenemos que podrá usar el valor que nosotros le asignemos o el de la plantilla. Si le asignamos un valor y queremos que este "machaque" el valor de la plantilla marcaremos la casilla que indica que sobrescribe la plantilla. Vemos aquí una imagen de la página de configuración de un servicio.

Figura 18: Página de configuración de un servicio de Nagiosadm

Cuando generemos los archivos de configuración tendremos que usar el usuario root o el usuario de nagios para poder reemplazar los archivos de configuración originales de la configuración y reiniciar Nagios para que tome la configuración que hemos generado.

PARTE VII : DOCUMENTACIÓN DEL CÓDIGO

13 Documentación del código

En los siguientes apartados podemos ver la estructuración de la aplicación ya explicada en puntos anteriores. Se ha preferido incluir el javadoc de la aplicación y no su código puesto que este es muy extenso y no tan práctico visualmente como la siguiente documentación en la que vamos viendo las funcionalidades de cada método, atributo y clase definido en la aplicación. En las capturas del javadoc que se muestran a continuación se han omitido las secciones de "Method Detail" en los casos en los que la clase implementaba una interfaz que ya tenía la explicación de los métodos y también los métodos que se heredan de la clase java.lang.Object.

No obstante, si el lector tiene necesidad de consultar el código de la aplicación, éste se encuentra incluido en el material que se entrega junto con esta memoria.

13.1 Paquetes

Packages	
nagios.actions	Paquete con las clases encargadas de la ejecución de cada una de las peticiones hechas a la aplicación
nagios.actions.interfaces	Interfaces que determinan el esquema de las clases que han de ejecutar las llamadas a la aplicación
nagios.CFG	Paquete de Gestión de los Ficheros de Configuración
nagios.CFG.file	Paquete con las clases que implementan los métodos de escritura y lectura de los ficheros de configuración
nagios.CFG.interfaces	Paquete con las interfaces de acceso a cada uno de los ficheros de configuración que ha de escribir o leer la aplicación para generar la configuración de Nagios.
nagios.ConfigurationObjects	Paquete con las clases que emulan los objetos de configuración de Nagios con sus atributos.
nagios.DAO	Paquete Data Access Object. Contiene toda la implementación relativa al acceso a base de datos.

nagios.DAO.interfaces	Paquete con las interfaces que definen los métodos de acceso para la inserción, actualización o borrado de los objetos de configuración en la base de datos.
nagios.DAO.oracle	Paquete con las clases que implementan las interfaces de acceso a la base de datos para el caso de una base de datos Oracle.
nagios.utilities	Paquete con clases que tienen constantes de la aplicación y usos varios, como la clase que es Filtro de la aplicación o el controlador de las llamadas realizadas a la aplicación.

13.2 Paquete nagios.actions.interfaces

Interface Summary	
Action	Interfaz con los métodos necesarios para tratar una acción relativa a un objeto de configuración de Nagios o para tratar el login del usuario.

13.2.1 Interfaz Action

public interface **Action**

Interfaz con los métodos necesarios para tratar una acción relativa a un objeto de configuración de Nagios o para tratar el login del usuario.

Method Summary	
`java.lang.String`	**`execute`**`(ReqUtil req, javax.servlet.http.HttpServletResponse res)` Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
`java.lang.Object`	**`getCommands`**`()`

	Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Method Detail

execute

```
java.lang.String execute(ReqUtil req,
                         javax.servlet.http.HttpServletResponse res)
```

```
                    throws javax.servlet.ServletException,
                           java.io.IOException
```

Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos. Los datos necesarios que se tengan que pasar como parámetros del request se introducirán en variables que serán devueltas por los métodos "get" de la clase.

Parameters:

`req` - Objeto ReqUtil con el que se obtendrán el valor de la acción y los valores que se hayan insertados en formularios de la aplicación.

`res` - Objeto HttpServletResponse para la redirección a otra URL.

Returns:

URL relativa de la aplicación a la que se ha de redireccionar o null en caso de que se no tenga que redireccionar a una dirección.

Throws:

`javax.servlet.ServletException`
`java.io.IOException`

---

getView

```
java.lang.String getView()
```

Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Returns:

URL a la que se redirecciona tras la ejecución de la acción.

---

getData

```
java.lang.Object getData()
```

Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.

Returns:

Datos relativos al objeto al que se refiere la acción.

---

getError

`java.lang.Object` **`getError`**`()`

Obtiene errores producidos en la ejecución de la acción.

Returns:
Posible error producido en la ejecución de la acción.

---

getResOperation

`java.lang.Object` **`getResOperation`**`()`

Returns:
Objeto resultado con el resultado de la operación. La forma en que se usará es indicando un String con el resultado y otro String con una dirección relativa de vuelta. Estos dos String se insertan en un Hashmap.

---

getTemplate

`java.lang.Object` **`getTemplate`**`()`

Obtiene la plantilla que usa el objeto.

Returns:
Plantilla que pueda usar el objeto al que se refiere la acción.

---

getHostgroups

`java.lang.Object` **`getHostgroups`**`()`

Obtiene todos los Hostgroups configurados.

Returns:
Hostgroups disponibles.

---

getTimeperiods

`java.lang.Object` **`getTimeperiods`**`()`

Obtiene todos los Timeperiods configurados.

Returns:
Timperiods disponibles.

---

getCommands

`java.lang.Object` **`getCommands`**`()`

Obtiene todos los Checkcommands configurados.

Returns:
Checkcommands disponibles.

---

getContactgroups

`java.lang.Object` **`getContactgroups`**`()`

Obtiene todos los contactgroups configurados.

Returns:
Contactgroups disponibles.

---

getPlntavaliables

`java.lang.Object` **`getPlntavaliables`**`()`

Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.

Returns:
Plantillas disponibles para el objeto al que se refiere la acción.

---

getContacts

`java.lang.Object` **`getContacts`**`()`

Obtiene todos los Contacts configurados.

Returns:
Contacts disponibles.

---

getHosts

`java.lang.Object` **`getHosts`**`()`

Obtiene todos los Hosts configurados.

Returns:
Hosts disponibles.

---

getServices

`java.lang.Object` **`getServices`**`()`

Obtiene todos los Services configurados.

Returns:
Services disponibles.

---

getPlatforms

`java.lang.Object` **`getPlatforms`**`()`

Obtiene todas las plataformas configuradas.

Returns:
Platforms disponibles.

13.3 Paquete nagios.actions

Class Summary

ActionFactory	Clase que crea las diferentes acciones que se pueden ejecutar.
CFGAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de archivos de configuración, las cuales son: cargarlos, escribirlos, recargar una configuración anterior y salvar la configuración actual en un archivo comprimido.
CommandAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Command, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
ContactAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contact, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
ContactgroupAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contactgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
DefaultAction	Clase que implementa los métodos de la interfaz Action para realizar la acción por defecto, que ser□ a la que se llame cuando la acción a realizar no coincida con ninguna de las existentes en FactoriaAccion
HostAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contactgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
HostDependencyAction	Clase encargada de llevar a cabo las acciones de gestión del objeto HostDependency.
HostEscalationAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Host Escalation.
HostgroupAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Hostgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
LoginAction	Clase que implementa los métodos de la interfaz Action para realizar el el login a la aplicación, mediante certificado digital
PlatformAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Platform, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.
ServiceAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Services, tanto para la presentación de los mismos

	al usuario de la aplicación como para su inserción o actualización en la base de data.
ServicegroupAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Servicegroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.
SrvDependencyAction	Clase encargada de llevar a cabo las acciones que gestionan los objetos Service Escalation.
SrvEscalationAction	Clase encargada de llevar a cabo la gestión de los objetos Service Escalation.
TemplateAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Template en todas sus versiones (ya que habrá plantillas de servicios, hosts y contactos), tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.
TimeperiodAction	Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Timeperiod, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.
UserAction	Clase que gestiona las acciones realizadas sobre los usuarios de administración de aplicación.

13.3.1 Clase ActionFactory

```
java.lang.Object
  └ nagios.actions.ActionFactory
```

```
public abstract class ActionFactory
extends java.lang.Object
```

Clase que crea las diferentes acciones que se pueden ejecutar.

Constructor Summary

ActionFactory()	

Method Summary	
static Action	**createAction**(java.lang.String action) Crea un objeto Action especifico.

Constructor Detail

ActionFactory

```
public ActionFactory()
```

Method Detail

createAction

```
public static Action createAction(java.lang.String action)
```

Crea un objeto Action especifico.

Parameters:
`action` - tipo de acción que se va a crear.

Returns:
un objeto Action que se encargara de realizar una determinada acción.

13.3.2 Clase CFGAction

```
java.lang.Object
  └nagios.actions.CFGAction
```

All Implemented Interfaces:
Action

```
public class CFGActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de archivos de configuración, las cuales son: cargarlos, escribirlos, recargar una configuración anterior y salvar la configuración actual en un archivo comprimido.

Author:
Agustín Bravo Ortiz

Constructor Summary

CFGAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.

java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

CFGAction

public **CFGAction**()

Constructor sin parámetros.

13.3.3 Clase CommandAction

java.lang.Object
  └ **nagios.actions.CommandAction**

All Implemented Interfaces:
Action

---

public class **CommandAction**extends java.lang.Objectimplements Action

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Command, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:
Agustín Bravo Ortiz

Constructor Summary

CommandAction() Constructor sin parámetros.	

Method Summary	
java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución

	en caso de que haya que redireccionar a alguna página.

Constructor Detail

CommandAction

```
public CommandAction()
```

Constructor sin parámetros.

13.3.4 Clase ContactAction

```
java.lang.Object
  └nagios.actions.ContactAction
```

All Implemented Interfaces:
Action

```
public class ContactActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contact, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:
Agustín Bravo Ortiz

Constructor Summary

ContactAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(RequUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la

	base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

ContactAction

```
public ContactAction()
```

Constructor sin parámetros.

13.3.5 Clase ContactgroupAction

```
java.lang.Object
  └nagios.actions.ContactgroupAction
```

All Implemented Interfaces:
Action

```
public class ContactgroupActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contactgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:
Agustín Bravo Ortiz

Constructor Summary

ContactgroupAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(RequUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.

java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

ContactgroupAction

```
public ContactgroupAction()
```

Constructor sin parámetros.

13.3.6 Clase DefaultAction

```
java.lang.Object
```

└ **nagios.actions.DefaultAction**

All Implemented Interfaces:
Action

---

public class **DefaultAction**extends java.lang.Objectimplements Action

Clase que implementa los métodos de la interfaz Accion para realizar la acción por defecto, que ser□ a la que se llame cuando la acción a realizar no coincida con ninguna de las existentes en FactoriaAccion

---

Constructor Summary

DefaultAction() Constructor de la clase.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**()

	Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

DefaultAction

public **DefaultAction**()

Constructor de la clase.

13.3.7 Clase HostAction

java.lang.Object
  └ **nagios.actions.HostAction**

All Implemented Interfaces:
Action

---

public class **HostAction**extends java.lang.Objectimplements Action

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Contactgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:
Agustín Bravo Ortiz

Constructor Summary

HostAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()

java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

HostAction

```
public HostAction()
```

Constructor sin parámetros.

13.3.8 Clase HostDependencyAction

```
java.lang.Object
  └nagios.actions.HostDependencyAction
```

All Implemented Interfaces:
Action

```
public class HostDependencyActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones de gestión del objeto HostDependency. El parámetro action es recogido de la llamada del usuario y las acciones posibles son añadir objetos, visualizar los objetos, borrarlos o cambiar sus parámetros. Cada acción recoge los parámetros necesarios de la base de datos o los inserta y reenvía al usuario a la página correspondiente.

Author:
Agustín Bravo Ortiz

Constructor Summary

HostDependencyAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**()

	Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

HostDependencyAction

```
public HostDependencyAction()
```

Constructor sin parámetros.

13.3.9 Clase HostEscalationAction

```
java.lang.Object
  └nagios.actions.HostEscalationAction
```

All Implemented Interfaces:
Action

```
public class HostEscalationActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Host Escalation. Según la acción podremos insertar nuevos objetos, actualizarlos, borrarlos o simplemente mostrarlos.

Author:
Agustín Bravo Ortiz

Constructor Summary

HostEscalationAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del

	nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

HostEscalationAction

```
public HostEscalationAction()
```

Constructor sin parámetros.

13.3.10 Clase HostgroupAction

```
java.lang.Object
  └nagios.actions.HostgroupAction
```

All Implemented Interfaces:
Action

```
public class HostgroupActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Hostgroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:
Agustín Bravo Ortiz

Constructor Summary	
HostgroupAction() Constructor sin parámetros.	

Method Summary	
java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.

java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

HostgroupAction

```
public HostgroupAction()
```

Constructor sin parámetros.

13.3.11 Clase LoginAction

```
java.lang.Object
  └ nagios.actions.LoginAction
```

All Implemented Interfaces:
Action

```
public class LoginActionextends java.lang.Objectimplements Action
```

Clase que implementa los métodos de la interfaz Accion para realizar el el login a la aplicación, mediante certificado digital

---

Constructor Summary

LoginAction() Constructor de la clase.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Ejecuta la accion inicializando la view y el modelo.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.

java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

LoginAction

public **LoginAction**()

Constructor de la clase.

13.3.12 Clase PlatformAction

java.lang.Object
  └ **nagios.actions.PlatformAction**

All Implemented Interfaces:
Action

---

public class **PlatformAction**extends java.lang.Objectimplements Action

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Platform, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:

Agustín Bravo Ortiz

Constructor Summary	
PlatformAction()	

Method Summary	
java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.

java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

PlatformAction

```
public PlatformAction()
```

13.3.13 Clase ServiceAction

```
java.lang.Object
  └nagios.actions.ServiceAction
```

All Implemented Interfaces:

Action

```
public class ServiceActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Services, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de data.

Author:

Agustín Bravo Ortiz

Constructor Summary

ServiceAction()	

Method Summary	
java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.

java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

ServiceAction

```
public ServiceAction()
```

13.3.14 Clase ServicegroupAction

```
java.lang.Object
  └nagios.actions.ServicegroupAction
```

All Implemented Interfaces:
Action

```
public class ServicegroupActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Servicegroups, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.

Author:
Agustín Bravo Ortiz

Constructor Summary

ServicegroupAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.

java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

ServicegroupAction

```
public ServicegroupAction()
```

Constructor sin parámetros.

13.3.15 Clase SrvDependencyAction

```
java.lang.Object
  └nagios.actions.SrvDependencyAction
```

All Implemented Interfaces:
Action

```
public class SrvDependencyActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones que gestionan los objetos Service Escalation. A partir del parámetro action controlamos las posibles acciones que se puedan llevar a cabo con estos objetos, como son inserciones de nuevos objetos, actualizaciones, borrado o presentación de los objetos al usuario de la aplicación.

Author:
Agustín Bravo Ortiz

Constructor Summary

SrvDependencyAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**()

	Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

SrvDependencyAction

```
public SrvDependencyAction()
```

Constructor sin parámetros.

13.3.16 Clase SrvEscalationAction

```
java.lang.Object
  └nagios.actions.SrvEscalationAction
```

All Implemented Interfaces:
Action

```
public class SrvEscalationActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo la gestión de los objetos Service Escalation. Esta clase maneja las acciones relativas a la muestra de los objetos al usuario, el borrado, su actualización, etc. La clase se encarga de según que acción se ejecuta, obtiene los objetos de la base de datos, o los procesa para su inserción (o borrado o actualización), carga los datos en el request de la petición web y redirecciona al usuario a la página adecuada.

Author:
Agustín Bravo Ortiz

Constructor Summary

SrvEscalationAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**()

	Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

SrvEscalationAction

```
public SrvEscalationAction()
```

Constructor sin parámetros.

13.3.17 Clase TemplateAction

```
java.lang.Object
  └ nagios.actions.TemplateAction
```

All Implemented Interfaces:
Action

```
public class TemplateActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Template en todas sus versiones (ya que habrá plantillas de servicios, hosts y contactos), tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.

Author:
Agustín Bravo Ortiz

Constructor Summary

TemplateAction() Constructor sin parámetros	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**()

	Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

TemplateAction

```
public TemplateAction()
```

Constructor sin parámetros

13.3.18 Clase TimeperiodAction

```
java.lang.Object
  └nagios.actions.TimeperiodAction
```

All Implemented Interfaces:
Action

```
public class TimeperiodActionextends java.lang.Objectimplements Action
```

Clase encargada de llevar a cabo las acciones relativas a la gestión de los objetos Timeperiod, tanto para la presentación de los mismos al usuario de la aplicación como para su inserción o actualización en la base de datos.

Author:
Agustín Bravo Ortiz

Constructor Summary

TimeperiodAction() Constructor sin parámetros.	

Method Summary

java.lang.String	**execute**(RequtIl req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.

java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

TimeperiodAction

```
public TimeperiodAction()
```

Constructor sin parámetros.

13.3.19 Clase UserAction

```
java.lang.Object
  └nagios.actions.UserAction
```

All Implemented Interfaces:
Action

```
public class UserActionextends java.lang.Objectimplements Action
```

Clase que gestiona las acciones realizadas sobre los usuarios de administración de aplicación. Se podrán insertar modificar o borrar.

Author:
Agustín Bravo Ortiz

Constructor Summary

UserAction() Constructor sin parámetros	

Method Summary

java.lang.String	**execute**(ReqUtil req, javax.servlet.http.HttpServletResponse res) Obtiene el tipo de acción que se ha de ejecutar y en función del

	nombre actuará para generar los datos necesarios obteniéndolos de la base de datos u obtener los datos de un formulario para insertarlos en la base de datos usando las clases de los objetos.
java.lang.Object	**getCommands**() Obtiene todos los Checkcommands configurados.
java.lang.Object	**getContactgroups**() Obtiene todos los contactgroups configurados.
java.lang.Object	**getContacts**() Obtiene todos los Contacts configurados.
java.lang.Object	**getData**() Obtiene datos relativos al objeto que se han obtenido durante la ejecución de la acción y son necesarios para la representación del mismo.
java.lang.Object	**getError**() Obtiene errores producidos en la ejecución de la acción.
java.lang.Object	**getHostgroups**() Obtiene todos los Hostgroups configurados.
java.lang.Object	**getHosts**() Obtiene todos los Hosts configurados.
java.lang.Object	**getPlatforms**() Obtiene todas las plataformas configuradas.
java.lang.Object	**getPlntavaliables**() Obtiene todas las plantillas configuradas y que puedan ser usadas con el objeto al que se refiere la acción.
java.lang.Object	**getResOperation**()
java.lang.Object	**getServices**() Obtiene todos los Services configurados.
java.lang.Object	**getTemplate**() Obtiene la plantilla que usa el objeto.
java.lang.Object	**getTimeperiods**() Obtiene todos los Timeperiods configurados.
java.lang.String	**getView**() Obtiene la vista a la que hay que redireccionar tras la ejecución en caso de que haya que redireccionar a alguna página.

Constructor Detail

UserAction

```
public UserAction()
```

Constructor sin parámetros

13.4 Paquete nagios.CFG

Class Summary	
CFGFactory	Clase encargada del acceso a los archivos de configuración.

13.4.1 Clase CFGFactory

```
java.lang.Object
  └nagios.CFG.CFGFactory
```

```
public class CFGFactoryextends java.lang.Object
```

Clase encargada del acceso a los archivos de configuración. Su utilidad será la de abrirlos, leer o escribir en ellos y posteriormente cerrarlos. Se usan bufferes de escritura o lectura para no tener un acceso continuo a los archivos.

Author:
Agustín Bravo Ortiz

Constructor Summary	
CFGFactory() Constructor sin parámetros.	

Method Summary	
void	closeFile() Cierra algún archivo que se haya abierto, ya sea en

	modo lectura o en modo escritura.
void	**flush**() Vuelca el contenido del buffer de escritura en el archivo.
java.io.BufferedReader	**openFileRd**(java.lang.String filename) Abre un archivo en modo lectura
void	**openFileWr**(java.lang.String filename) Abre un archivo en modo escritura.
java.lang.String	**readLine**() Método para leer una linea útil de un archivo.
void	**write**(java.lang.String to) Escribe en el buffer de escritura del archivo.

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

CFGFactory

```
public CFGFactory()
```

Constructor sin parámetros.

Method Detail

openFileRd

```
public java.io.BufferedReader openFileRd(java.lang.String filename)
                                  throws java.io.FileNotFoundException
```

Abre un archivo en modo lectura

Parameters:
`filename` - Nombre con path completo del archivo que se quiere abrir.
Returns:
BufferedReader para escritura del fichero.
Throws:
`java.io.FileNotFoundException`

readLine

```
public java.lang.String readLine()
                         throws java.io.IOException
```

Método para leer una linea útil de un archivo. Se omiten lineas en blanco y comentarios. En caso de tener lineas útiles con comentarios al final se devolverá sin este comentario. Las comillas son "escapadas" para poder insertar la cadena en la base de datos.

Returns:
String con la linea leída.
Throws:
`java.io.IOException`

---

openFileWr

```
public void openFileWr(java.lang.String filename)
               throws java.io.IOException
```

Abre un archivo en modo escritura.

Parameters:
`filename` - Path completo del archivo que se quiere abrir.
Throws:
`java.io.IOException`

---

write

```
public void write(java.lang.String to)
          throws java.io.IOException
```

Escribe en el buffer de escritura del archivo.

Parameters:
`to` - String con el contenido de lo que se va a escribir en el archivo.
Throws:
`java.io.IOException`

---

flush

```
public void flush()
           throws java.io.IOException
```

Vuelca el contenido del buffer de escritura en el archivo.

Throws:
`java.io.IOException`

---

closeFile

```
public void closeFile()
               throws java.io.IOException
```

Cierra algún archivo que se haya abierto, ya sea en modo lectura o en modo escritura.

Throws:
`java.io.IOException`

13.5 Paquete nagios.CFG.interfaces

Interface Summary	
CommandCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Command adecuada a la documentación de Nagios 2.5
ContactCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Contact adecuada a la documentación de Nagios 2.5
ContactgroupCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Contactgroup adecuada a la documentación de Nagios 2.5
HostCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Host adecuada a la documentación de Nagios 2.5
HostgroupCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Hostgroup adecuada a la documentación de Nagios 2.5

ServiceCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Service adecuada a la documentación de Nagios 2.5
ServicegroupCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Servicegroup adecuada a la documentación de Nagios 2.5
TimeperiodCFG	Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Timeperiod adecuada a la documentación de Nagios 2.5

13.5.1 Interfaz CommandCFG

```
public interface CommandCFG
```

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Command adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary

CommandTO[]	**readCommands**(Result result) Método para leer de un archivo la configuración de comandos de chequeo de Nagios.
Result	**writeCommands**(CommandTO[] to) Método para escribir el archivo de configuración del objeto CommandTO.

Method Detail

writeCommands

```
Result writeCommands(CommandTO[] to)
```

Método para escribir el archivo de configuración del objeto CommandTO.

Parameters:
`to` - Array con los objetos CommandTO para escribirlos
Returns:

Resultado de la acción

---

readCommands

`CommandTO[] readCommands(Result result)`

Método para leer de un archivo la configuración de comandos de chequeo de Nagios.

Parameters:
`result` - Objeto con el resultado de realizar la operación
Returns:
CommandTO con los comandos configurados

13.5.2 Interfaz ContactCFG

`public interface ContactCFG`

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Contact adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary	
ContactTO[]	**readContacts**(Result result) Método para leer de un archivo de configuración los contactos configurados
Result	**writeContacts**(ContactTO[] to) Escribe en el archivo de configuración la configuración correspondiente a los comandos que se le pasan.

Method Detail

readContacts

`ContactTO[] readContacts(Result result)`

Método para leer de un archivo de configuración los contactos configurados

Parameters:
`result` - Resultado de la operación, es pasado por referencia.
Returns:
ContactTO[] array de contactos con la configuración

---

writeContacts

`Result` **`writeContacts`**`(ContactTO[] to)`

Escribe en el archivo de configuración la configuración correspondiente a los comandos que se le pasan.

Parameters:
`to` - Array con los comandos que han de ser configurados
Returns:
Resultado de la acción

13.5.3 Interfaz ContactgroupCFG

`public interface` **`ContactgroupCFG`**

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Contactgroup adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary	
`ContactgroupTO[]`	**`readContactgroups`**`(Result result)` Método para recoger la configuración de los grupos de contactos a partir de un archivo de configuración
`Result`	**`writeContactgroups`**`(ContactgroupTO[] to)` Escribe en el archivo la configuración correspondiente a los objetos contactgroups que se le pasan.

Method Detail

writeContactgroups

`Result` **`writeContactgroups`**`(ContactgroupTO[] to)`

Escribe en el archivo la configuración correspondiente a los objetos contactgroups que se le pasan.

Parameters:
`to` - Array con los Contactgroups que han de ser configurados.
Returns:
Resultado de la operación.

---

readContactgroups

`ContactgroupTO[]` **`readContactgroups`**`(Result result)`

Método para recoger la configuración de los grupos de contactos a partir de un archivo de configuración

Parameters:
`result` - Resultado de la operación. El objeto es pasado por referencia
Returns:
Array con los Contactgroups que han sido leídos del archivo de configuración.

13.5.4 Interfaz HostCFG

public interface HostCFG

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Host adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary	
`HostTO[]`	**`readHosts`**`(Result result)` Método para recoger la configuración de los Hosts a partir de un archivo de configuración
`Result`	**`writeHosts`**`(HostTO[] to)`

	Escribe en el archivo la configuración correspondiente a los objetos Hosts que se le pasan.

Method Detail

writeHosts

`Result` **`writeHosts`**`(HostTO[] to)`

Escribe en el archivo la configuración correspondiente a los objetos Hosts que se le pasan.

Parameters:
`to` - Array con los Hosts que han de ser configurados.
Returns:
Resultado de la operación.

---

readHosts

`HostTO[]` **`readHosts`**`(Result result)`

Método para recoger la configuración de los Hosts a partir de un archivo de configuración

Parameters:
`result` - Resultado de la operación. Es pasado por referencia.
Returns:
Array con los objetos Hosts que han sido leídos del archivo de configuración.

13.5.5 Interfaz HostgroupCFG

`public interface` **`HostgroupCFG`**

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Hostgroup adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary	
HostgroupTO[]	**readHostgroups**(Result result) Método para recoger la configuración de los grupos de hosts de un archivo de configuración.
Result	**writeHostgroups**(HostgroupTO[] to) Escribe en el archivo la configuración correspondiente a los objetos Hostgroups que se le pasan.

Method Detail

writeHostgroups

Result **writeHostgroups**(HostgroupTO[] to)

Escribe en el archivo la configuración correspondiente a los objetos Hostgroups que se le pasan.

Parameters:
`to` - Array con los Hostgroups que han de ser configurados
Returns:
Resultado de la operación

---

readHostgroups

HostgroupTO[] **readHostgroups**(Result result)

Método para recoger la configuración de los grupos de hosts de un archivo de configuración.

Parameters:
`result` - Resultado de la operación. Está pasado por referencia.
Returns:
Array con los Hostgroups que han sido leídos del archivo de configuración.

13.5.6 Interfaz ServiceCFG

```
public interface ServiceCFG
```

Interfaz con los métodos necesarios para la lectura y escritura de la configuración

correspondiente a los objetos Service adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary	
ServiceTO[]	**readServices**(Result result) Método para recoger la configuracion de los servicios desde un archivo de configuración
Result	**writeServices**(ServiceTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Services.

Method Detail

writeServices

Result **writeServices**(ServiceTO[] to)

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Services.

Parameters:
`to` - Array con los Services que han de ser escritos en el archivo de configuración.
Returns:
Resultado de la operación.

---

readServices

ServiceTO[] **readServices**(Result result)

Método para recoger la configuracion de los servicios desde un archivo de configuración

Parameters:
`result` - Resultado de la operación. Está pasado por referencia.
Returns:
Array con los Services que han sido leídos del archivo de configuración.

13.5.7 Interfaz ServicegroupCFG

```
public interface ServicegroupCFG
```

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Servicegroup adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary

ServicegroupTO[]	**readServicegroups**(Result result) Método para recoger la configuración de los grupos de servicios de un archivo de configuración.
Result	**writeServicegroups**(ServicegroupTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Servicegroups.

Method Detail

writeServicegroups

```
Result writeServicegroups(ServicegroupTO[] to)
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Servicegroups.

Parameters:
`to` - Array con los Servicegroups que han de ser escritos en el archivo de configuración.
Returns:
Resultado de la operación.

---

readServicegroups

```
ServicegroupTO[] readServicegroups(Result result)
```

Método para recoger la configuración de los grupos de servicios de un archivo de

configuración.

Parameters:
`result` - Resultado de la operación. Está pasado por referencia.
Returns:
Array con los Servicegroups que han sido leídos del archivo de configuración.

13.5.8 Interfaz TimeperiodCFG

```
public interface TimeperiodCFG
```

Interfaz con los métodos necesarios para la lectura y escritura de la configuración correspondiente a los objetos Timeperiod adecuada a la documentación de Nagios 2.5

Author:
Agustín Bravo Ortiz

Method Summary

TimeperiodTO[]	**readTimeperiods**(Result result) Método para recoger la configuración de los timeperiods de un archivo de configuración.
Result	**writeTimeperiods**(TimeperiodTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Timeperiods.

Method Detail

writeTimeperiods

```
Result writeTimeperiods(TimeperiodTO[] to)
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Timeperiods.

Parameters:
`to` - Array con los Timeperiods que han de ser escritos en el archivo de configuración.
Returns:
Resultado de la operación.

readTimeperiods

`TimeperiodTO[] readTimeperiods(Result result)`

Método para recoger la configuración de los timeperiods de un archivo de configuración.

Parameters:
`result` - Resultado de la operación. Está pasado por referencia.

Returns:
Array con los Timeperiods que han sido leídos del archivo de configuración.

13.6 Paquete nagios.CFG.file

Class Summary	
FileCommandCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los comandos de chequeo y notificación de la aplicación.
FileContactCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los contactos de la aplicación.
FileContactgroupCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Contactgroups.
FileHostCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Hosts.
FileHostgroupCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Hostgroups.
FileServiceCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Services.
FileServicegroupCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Servicegroups.
FileTimeperiodCFG	Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Timeperiods.

13.6.1 Clase FileCommandCFG

```
java.lang.Object
  └ nagios.CFG.file.FileCommandCFG
```

All Implemented Interfaces:
CommandCFG

```
public class FileCommandCFGextends java.lang.Objectimplements CommandCFG
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los comandos de chequeo y notificación de la aplicación.

Author:
Agustín Bravo Ortiz

Constructor Summary

FileCommandCFG()	

Method Summary

CommandTO[]	**readCommands**(Result result) Método para leer de un archivo la configuración de comandos de chequeo de Nagios.
Result	**writeCommands**(CommandTO[] to) Método para escribir el archivo de configuración del objeto CommandTO.

Constructor Detail

FileCommandCFG

```
public FileCommandCFG()
```

13.6.2 Clase FileContactCFG

```
java.lang.Object
```

└ **nagios.CFG.file.FileContactCFG**

All Implemented Interfaces:
ContactCFG

---

public class **FileContactCFG**extends java.lang.Objectimplements ContactCFG

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los contactos de la aplicación.

Author:
Agustín Bravo Ortiz

Constructor Summary

FileContactCFG()	

Method Summary

ContactTO[]	**readContacts**(Result result) Método para leer de un archivo de configuración los contactos configurados
Result	**writeContacts**(ContactTO[] to) Escribe en el archivo de configuración la configuración correspondiente a los comandos que se le pasan.

Constructor Detail

FileContactCFG

public **FileContactCFG**()

13.6.3 Clase FileContactgroupCFG

java.lang.Object
└ **nagios.CFG.file.FileContactgroupCFG**

All Implemented Interfaces:

ContactgroupCFG

---

public class **FileContactgroupCFG**extends java.lang.Objectimplements ContactgroupCFG

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Contactgroups.

Author:
Agustín Bravo Ortiz

Constructor Summary

FileContactgroupCFG()	

Method Summary

ContactgroupTO[]	**readContactgroups**(Result result) Método para recoger la configuración de los grupos de contactos a partir de un archivo de configuración
Result	**writeContactgroups**(ContactgroupTO[] to) Escribe en el archivo la configuración correspondiente a los objetos contactgroups que se le pasan.

Constructor Detail

FileContactgroupCFG

public **FileContactgroupCFG**()

13.6.4 Clase FileHostCFG

java.lang.Object
  └ **nagios.CFG.file.FileHostCFG**

All Implemented Interfaces:
HostCFG

public class **FileHostCFG**extends java.lang.Objectimplements HostCFG

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Hosts.

Author:
Agustín Bravo Ortiz

Constructor Summary

FileHostCFG()	

Method Summary

HostTO[]	**readHosts**(Result result) Método para recoger la configuración de los Hosts a partir de un archivo de configuración
Result	**writeHosts**(HostTO[] to) Escribe en el archivo la configuración correspondiente a los objetos Hosts que se le pasan.

Constructor Detail

FileHostCFG

public **FileHostCFG**()

13.6.5 Clase FileHostgroupCFG

java.lang.Object
  └ **nagios.CFG.file.FileHostgroupCFG**

All Implemented Interfaces:
HostgroupCFG

public class **FileHostgroupCFG**extends java.lang.Objectimplements HostgroupCFG

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Hostgroups.

Author:
Agustín Bravo Ortiz

Constructor Summary	
FileHostgroupCFG()	

Method Summary	
HostgroupTO[]	**readHostgroups**(Result result) Método para recoger la configuración de los grupos de hosts de un archivo de configuración.
Result	**writeHostgroups**(HostgroupTO[] to) Escribe en el archivo la configuración correspondiente a los objetos Hostgroups que se le pasan.

Constructor Detail

FileHostgroupCFG

```
public FileHostgroupCFG()
```

13.6.6 Clase FileServiceCFG

```
java.lang.Object
  └nagios.CFG.file.FileServiceCFG
```

All Implemented Interfaces:
ServiceCFG

```
public class FileServiceCFGextends java.lang.Objectimplements ServiceCFG
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Services.

Author:
Agustín Bravo Ortiz

Constructor Summary	
FileServiceCFG()	

Method Summary	
ServiceTO[]	**readServices**(Result result) Método para recoger la configuracion de los servicios desde un archivo de configuración
Result	**writeServices**(ServiceTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Services.

Constructor Detail

FileServiceCFG

```
public FileServiceCFG()
```

13.6.7 Clase FileServicegroupCFG

```
java.lang.Object
  └ nagios.CFG.file.FileServicegroupCFG
```

All Implemented Interfaces:
ServicegroupCFG

```
public class FileServicegroupCFGextends java.lang.Objectimplements
ServicegroupCFG
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Servicegroups.

Author:
Agustín Bravo Ortiz

Constructor Summary	
FileServicegroupCFG()	

Method Summary	
ServicegroupTO[]	**readServicegroups**(Result result) Método para recoger la configuración de los grupos de servicios de un archivo de configuración.
Result	**writeServicegroups**(ServicegroupTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Servicegroups.

Constructor Detail

FileServicegroupCFG

```
public FileServicegroupCFG()
```

13.6.8 Clase FileTimeperiodCFG

```
java.lang.Object
  └nagios.CFG.file.FileTimeperiodCFG
```

All Implemented Interfaces:

TimeperiodCFG

```
public class FileTimeperiodCFGextends java.lang.Objectimplements
TimeperiodCFG
```

Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Timeperiods.

Author:

Agustín Bravo Ortiz

Constructor Summary	
FileTimeperiodCFG()	

Method Summary	
TimeperiodTO[]	**readTimeperiods**(Result result) Método para recoger la configuración de los timeperiods de un archivo de configuración.
Result	**writeTimeperiods**(TimeperiodTO[] to) Clase encargada del acceso de escritura y lectura de los archivos de configuración que contienen los Timeperiods.

Constructor Detail

FileTimeperiodCFG

```
public FileTimeperiodCFG()
```

13.7 Paquete nagios.ConfigurationObjects

Class Summary	
AdmUserTO	Esta clase contiene un objeto de usuario administrador y los métodos para obtener y fijar sus variables.
CommandTO	Clase que representa el objeto de configuración Command definido en Nagios 2.5.
ContactgroupTO	Clase que representa el objeto de configuración Contactgroup definido en Nagios 2.5.
ContactTO	Clase que representa el objeto de configuración Contact definido en Nagios 2.5.
HostDependencyTO	Clase que representa el objeto de configuración Hostdependenc definido en Nagios 2.5.
HostEscalationTO	Clase que representa el objeto de configuración Hostescalation definido en Nagios 2.5.

HostgroupTO	Clase que representa el objeto de configuración Hostgroup definido en Nagios 2.5.
HostTO	Clase que representa el objeto de configuración Host definido en Nagios 2.5.
PlatformTO	Este objeto representa un grupo y es usado para el alamcenamiento y gestión de plataformas en la base de datos.
ServicegroupTO	Clase que representa el objeto de configuración Servicegroup definido en Nagios 2.5.
ServiceTO	Clase que representa el objeto de configuración Service definido en Nagios 2.5.
SrvDependencyTO	Clase que representa el objeto de configuración Servicedependency definido en Nagios 2.5.
SrvEscalationTO	Clase que representa el objeto de configuración Serviceescalation definido en Nagios 2.5.
TimeperiodTO	Clase que representa el objeto de configuración Timeperiod definido en Nagios 2.5.

13.7.1 Clase AdmUserTO

```
java.lang.Object
  └nagios.ConfigurationObjects.AdmUserTO
```

```
public class AdmUserTOextends java.lang.Object
```

Esta clase contiene un objeto de usuario administrador y los métodos para obtener y fijar sus variables. Serán las mismas que se guardan en la tabla de administradores de la aplicación de nagios.

Author:
Agustín Bravo Ortiz

Constructor Summary

AdmUserTO() Constructor sin parámetros	

Method Summary

java.lang.String	**getEmail**()
boolean	**getEstado**()
java.util.Date	**getFecuce**()
java.lang.String	**getName**()
java.lang.String	**getNif**()
java.lang.String	**getPassword**()
boolean	**isActivo**()
boolean	**isAdmin**()
void	**setAdmin**(boolean admin)
void	**setEmail**(java.lang.String email)
void	**setEstado**(boolean estado)
void	**setFecuce**(java.util.Date fecuce)
void	**setName**(java.lang.String name)
void	**setNif**(java.lang.String nif)
void	**setPassword**(java.lang.String password)

Constructor Detail

AdmUserTO

public **AdmUserTO**()

Constructor sin parámetros

13.7.2 Clase CommandTO

```
java.lang.Object
  └ nagios.ConfigurationObjects.CommandTO
```

```
public class CommandTOextends java.lang.Object
```

Clase que representa el objeto de configuración Command definido en Nagios 2.5. Este objeto representa un comando de chequeo de servicios o de notificación a los usuarios del servicio Nagios.

Author:
Agustín Bravo Ortiz

Constructor Summary	
CommandTO()	

Method Summary	
java.lang.String	getCommand_line()
java.lang.String	getCommand_name()
boolean	isNotifica()
void	setCommand_line(java.lang.String command_line)
void	setCommand_name(java.lang.String command_name)
void	setNotifica(boolean notifica)
java.lang.String	toString()

Constructor Detail

CommandTO

```
public CommandTO()
```

13.7.3 Clase ContactgroupTO

```
java.lang.Object
  └nagios.ConfigurationObjects.ContactgroupTO
```

```
public class ContactgroupTOextends java.lang.Object
```

Clase que representa el objeto de configuración Contactgroup definido en Nagios 2.5. Este objeto representa un grupo de contactos de Nagios.

Author:
Agustín Bravo Ortiz

Constructor Summary

ContactgroupTO()	

Method Summary

java.lang.String	getAlias()
java.lang.String	getContactgroup_name()
java.lang.String[]	getMembers()
java.lang.String	getPlatform()
void	setAlias(java.lang.String alias)
void	setContactgroup_name(java.lang.String contactgroup_name)

void	**setMembers**(java.lang.String[] members)
void	**setPlatform**(java.lang.String platform)
java.lang.String	**toString**()

Constructor Detail

ContactgroupTO

public **ContactgroupTO**()

13.7.4 Clase ContactTO

java.lang.Object
  └ **nagios.ConfigurationObjects.ContactTO**

---

public class **ContactTO**extends java.lang.Object

Clase que representa el objeto de configuración Contact definido en Nagios 2.5. Este objeto representa un usuario del servicio Nagios.

Author:
Agustín Bravo Ortiz

Constructor Summary

ContactTO()	

Method Summary

Trait	**getAddressx**()
Trait	**getAlias**()

Trait	**getContact_name**()
Trait[]	**getContactgroups**()
Trait	**getEmail**()
Trait	**getHost_notification_commands**()
Trait	**getHost_notification_options**()
Trait	**getHost_notification_period**()
Trait	**getId_Contact**()
Trait	**getName**()
Trait	**getPager**()
Trait	**getPlatform**()
Trait	**getRegister**()
Trait	**getService_notification_commands**()
Trait	**getService_notification_options**()
Trait	**getService_notification_period**()
Trait	**getTemplate**()
void	**setAddressx**(Trait addressx)
void	**setAlias**(Trait alias)
void	**setContact_name**(Trait contact_name)

void	**setContactgroups**(Trait[] contactgroups)
void	**setEmail**(Trait email)
void	**setHost_notification_commands**(Trait host_notification_commands)
void	**setHost_notification_options**(Trait host_notification_options)
void	**setHost_notification_period**(Trait host_notification_period)
void	**setId_Contact**(Trait id_Contact)
void	**setName**(Trait name)
void	**setPager**(Trait pager)
void	**setPlatform**(Trait plataforma)
void	**setRegister**(Trait register)
void	**setService_notification_commands**(Trait service_notification_commands)
void	**setService_notification_options**(Trait service_notification_options)
void	**setService_notification_period**(Trait service_notification_period)
void	**setTemplate**(Trait plantilla)
java.lang.String	**toString**()

Constructor Detail

ContactTO

```
public ContactTO()
```

13.7.5 Clase HostgroupTO

```
java.lang.Object
  └nagios.ConfigurationObjects.HostgroupTO
```

```
public class HostgroupTOextends java.lang.Object
```

Clase que representa el objeto de configuración Hostgroup definido en Nagios 2.5. Este objeto representa un grupo de Host.

Author:
Agustín Bravo Ortiz

Constructor Summary	
HostgroupTO()	

Method Summary	
java.lang.String	getAlias()
java.lang.String	getHostgroup_name()
java.lang.String[]	getMembers()
java.lang.String	getPlatform()
void	setAlias(java.lang.String alias)
void	setHostgroup_name(java.lang.String hostgroup_name)
void	setMembers(java.lang.String[] members)

void	**setPlatform**(java.lang.String platform)
java.lang.String	**toString**()

Constructor Detail

HostgroupTO

public **HostgroupTO**()

13.7.6 Clase HostTO

java.lang.Object
  └ **nagios.ConfigurationObjects.HostTO**

---

public class **HostTO**extends java.lang.Object

Clase que representa el objeto de configuración Host definido en Nagios 2.5. Este objeto representa un servidor que es chequeado periódicamente por Nagios.

Author:
Agustín Bravo Ortiz

Constructor Summary

HostTO() Constructor sin argumentos	

Method Summary

void	**addContact_groups**(Trait contact_groups) Añade un grupo de contactos al atributo contact_groups del Host.
void	**addHostgroups**(Trait hostgroups) Añade grupo de hosts al parámetro hostgroups del Host.

void	**addParents**(Trait parents) Añade un parámetro más al atributo parents
Trait	**getAction_url**()
Trait	**getActive_checks_enabled**()
Trait	**getAddress**()
Trait	**getAlias**()
Trait	**getCheck_command**()
Trait	**getCheck_freshness**()
Trait	**getCheck_interval**()
Trait	**getCheck_period**()
Trait[]	**getContact_groups**()
Trait	**getDosd_coords**()
Trait	**getEvent_handler_enabled**()
Trait	**getEvent_handler**()
Trait	**getFlap_detection_enabled**()
Trait	**getFreshness_threshold**()
Trait	**getHigh_flap_threshold**()
Trait	**getHost_name**()
Trait[]	**getHostgroups**()
Trait	**getIcon_image_alt**()

Trait	**getIcon_image**()
Trait	**getId_Host**()
Trait	**getLow_flap_threshold**()
Trait	**getMax_check_attempts**()
Trait	**getName**()
Trait	**getNotes_url**()
Trait	**getNotes**()
Trait	**getNotification_interval**()
Trait	**getNotification_options**()
Trait	**getNotification_period**()
Trait	**getNotifications_enabled**()
Trait	**getObsess_over_host**()
Trait[]	**getParents**()
Trait	**getPassive_checks_enabled**()
Trait	**getPlatform**()
Trait	**getProcess_perf_data**()
Trait	**getRegister**()
Trait	**getRetain_nonstatus_information**()

Trait	**getRetain_status_information**()
Trait	**getStalking_options**()
Trait	**getStatusmap_image**()
Trait	**getTemplate**()
Trait	**getTresd_coords**()
Trait	**getVrml_image**()
Trait	**isActive_checks_enabled**()
Trait	**isCheck_freshness**()
Trait	**isEvent_handler_enabled**()
Trait	**isFlap_detection_enabled**()
Trait	**isNotifications_enabled**()
Trait	**isObsess_over_host**()
Trait	**isPassive_checks_enabled**()
Trait	**isProcess_perf_data**()
Trait	**isRetain_nonstatus_information**()
Trait	**isRetain_status_information**()
void	**setAction_url**(Trait action_url)
void	**setActive_checks_enabled**(Trait active_checks_enabled)

void	**setAddress**(Trait address)
void	**setAlias**(Trait alias)
void	**setCheck_command**(Trait checkcommand)
void	**setCheck_freshness**(Trait check_freshness)
void	**setCheck_interval**(Trait check_interval)
void	**setCheck_period**(Trait check_period)
void	**setContact_groups**(Trait[] contact_groups)
void	**setDosd_coords**(Trait dosd_coords)
void	**setEvent_handler_enabled**(Trait event_handler_enabled)
void	**setEvent_handler**(Trait event_handler)
void	**setFlap_detection_enabled**(Trait flap_detection_enabled)
void	**setFreshness_threshold**(Trait freshness_threshold)
void	**setHigh_flap_threshold**(Trait high_flap_threshold)
void	**setHost_name**(Trait host_name)
void	**setHostgroups**(Trait[] hostgroups)
void	**setIcon_image_alt**(Trait icon_image_alt)
void	**setIcon_image**(Trait icon_image)

void	**setId_Host**(Trait id_Host)
void	**setLow_flap_threshold**(Trait low_flap_threshold)
void	**setMax_check_attempts**(Trait max_check_attempts)
void	**setName**(Trait name)
void	**setNotes_url**(Trait notes_url)
void	**setNotes**(Trait notes)
void	**setNotification_interval**(Trait notification_interval)
void	**setNotification_options**(Trait notification_options)
void	**setNotification_period**(Trait notification_period)
void	**setNotifications_enabled**(Trait notifications_enabled)
void	**setObsess_over_host**(Trait obsess_over_host)
void	**setParents**(Trait[] parents)
void	**setPassive_checks_enabled**(Trait passive_checks_enabled)
void	**setPlatform**(Trait plataforma)
void	**setProcess_perf_data**(Trait process_perf_data)
void	**setRegister**(Trait register)

void	**setRetain_nonstatus_information**(Trait retain_nonstatus_information)
void	**setRetain_status_information**(Trait retain_status_information)
void	**setStalking_options**(Trait stalking_options)
void	**setStatusmap_image**(Trait statusmap_image)
void	**setTemplate**(Trait plantilla)
void	**setTresd_coords**(Trait tresd_coords)
void	**setVrml_image**(Trait vrml_image)
java.lang.String	**toString**()

Constructor Detail

HostTO

public **HostTO**()

Constructor sin argumentos

13.7.7 Clase PlatformTO

java.lang.Object
  └ **nagios.ConfigurationObjects.PlatformTO**

---

public class **PlatformTO**extends java.lang.Object

Este objeto representa un grupo y es usado para el almacenamiento y gestión de plataformas en la base de datos. Los Host y otros servicios estarán agrupados según plataformas.

Author:

Agustín Bravo Ortiz

Constructor Summary	
PlatformTO()	

Method Summary	
java.lang.String	**getPlatform_name**()
void	**setPlatform_name**(java.lang.String platform_name)

Constructor Detail

PlatformTO

public **PlatformTO**()

13.7.8 Clase ServicegroupTO

java.lang.Object
  └ **nagios.ConfigurationObjects.ServicegroupTO**

---

public class **ServicegroupTO**extends java.lang.Object

Clase que representa el objeto de configuración Servicegroup definido en Nagios 2.5. Este objeto representa un grupo de servicios y su utilidad es la de que Nagios pueda representar información conjunta sobre los mismos.

Author:
Agustín Bravo Ortiz

Constructor Summary

ServicegroupTO()	

Method Summary	
java.lang.String	**getAlias**()
ServiceTO[]	**getMembers**()
java.lang.String	**getPlatform**()
java.lang.String	**getServicegroup_name**()
void	**setAlias**(java.lang.String alias)
void	**setMembers**(ServiceTO[] members)
void	**setPlatform**(java.lang.String platform)
void	**setServicegroup_name**(java.lang.String servicegroup_name)
java.lang.String	**toString**()

Constructor Detail

ServicegroupTO

public **ServicegroupTO**()

13.7.9 Clase ServiceTO

java.lang.Object
  └ **nagios.ConfigurationObjects.ServiceTO**

---

public class **ServiceTO**extends java.lang.Object

Clase que representa el objeto de configuración Service definido en Nagios 2.5. Este objeto representa un servicio que será chequeado por Nagios en base a su configuración.

Author:
Agustín Bravo Ortiz

Constructor Summary

ServiceTO() Constructor sin parámetros.	

Method Summary

void	**addContact_groups**(Trait contact_groups) Añade un grupo de contactos al atrubuto
void	**addServicegroups**(Trait servicegroups) Añade un grupo de servicios
ServiceTO	**clone**(ServiceTO srv) Crea un objeto idéntico con los mismos parámetros
Trait	**getAction_url**()
Trait	**getActive_checks_enabled**()
Trait	**getCheck_command**()
Trait	**getCheck_freshness**()
Trait	**getCheck_period**()
Trait	**getCommand_arguments**()
Trait[]	**getContact_groups**()
Trait	**getEvent_handler_enabled**()
Trait	**getEvent_handler**()
Trait	**getFlap_detection_enabled**()

Trait	**getFreshness_threshold**()
Trait	**getHigh_flap_threshold**()
Trait	**getHost_name**()
Trait	**getIcon_image_alt**()
Trait	**getIcon_image**()
Trait	**getId_Service**()
Trait	**getIs_volatile**()
Trait	**getLow_flap_threshold**()
Trait	**getMax_check_attempts**()
Trait	**getName**()
Trait	**getNormal_check_interval**()
Trait	**getNotes_url**()
Trait	**getNotes**()
Trait	**getNotification_interval**()
Trait	**getNotification_options**()
Trait	**getNotification_period**()
Trait	**getNotifications_enabled**()
Trait	**getObsess_over_service**()
Trait	**getParallelize_check**()

Trait	**getPassive_checks_enabled**()
Trait	**getPlatform**()
Trait	**getProcess_perf_data**()
Trait	**getRegister**()
Trait	**getRetain_nonstatus_information**()
Trait	**getRetain_status_information**()
Trait	**getRetry_check_interval**()
Trait	**getService_description**()
Trait[]	**getServicegroups**()
Trait	**getStalking_options**()
Trait	**getTemplate**()
void	**setAction_url**(Trait action_url)
void	**setActive_checks_enabled**(Trait active_checks_enabled)
void	**setCheck_command**(Trait check_command)
void	**setCheck_freshness**(Trait check_freshness)
void	**setCheck_period**(Trait check_period)
void	**setCommand_arguments**(Trait command_arguments)
void	**setContact_groups**(Trait[] contact_groups)

void	**setEvent_handler_enabled**(Trait event_handler_enabled)
void	**setEvent_handler**(Trait event_handler)
void	**setFlap_detection_enabled**(Trait flap_detection_enabled)
void	**setFreshness_threshold**(Trait freshness_threshold)
void	**setHigh_flap_threshold**(Trait high_flap_threshold)
void	**setHost_name**(Trait host_name)
void	**setIcon_image_alt**(Trait icon_image_alt)
void	**setIcon_image**(Trait icon_image)
void	**setId_Service**(Trait Id_Service)
void	**setIs_volatile**(Trait is_volatile)
void	**setLow_flap_threshold**(Trait low_flap_threshold)
void	**setMax_check_attempts**(Trait max_check_attempts)
void	**setName**(Trait name)
void	**setNormal_check_interval**(Trait normal_check_interval)
void	**setNotes_url**(Trait notes_url)

void	**setNotes**(Trait notes)
void	**setNotification_interval**(Trait notification_interval)
void	**setNotification_options**(Trait notification_options)
void	**setNotification_period**(Trait notification_period)
void	**setNotifications_enabled**(Trait notifications_enabled)
void	**setObsess_over_service**(Trait obsess_over_service)
void	**setParallelize_check**(Trait parallelize_check)
void	**setPassive_checks_enabled**(Trait passive_checks_enabled)
void	**setPlatform**(Trait plataforma)
void	**setProcess_perf_data**(Trait process_perf_data)
void	**setRegister**(Trait register)
void	**setRetain_nonstatus_information**(Trait retain_nonstatus_information)
void	**setRetain_status_information**(Trait retain_status_information)
void	**setRetry_check_interval**(Trait retry_check_interval)
void	**setService_description**(Trait service_description)

void	**setServicegroups**(Trait[] servicegroups)
void	**setStalking_options**(Trait stalking_options)
void	**setTemplate**(Trait plantilla)
java.lang.String	**toString**()

Constructor Detail

ServiceTO

public **ServiceTO**()

Constructor sin parámetros.

13.7.10 Clase TimeperiodTO

java.lang.Object
  └ **nagios.ConfigurationObjects.TimeperiodTO**

---

public class **TimeperiodTO**extends java.lang.Object

Clase que representa el objeto de configuración Timeperiod definido en Nagios 2.5. Este objeto representa un periodo de tiempo por el que se chequeará un servicio, un host o por el que se notifica a un administrador, etc.

Author:
Agustín Bravo Ortiz

Constructor Summary

TimeperiodTO()	

Method Summary	
java.lang.String	**getAlias**()
java.lang.String	**getFriday**()
java.lang.String	**getMonday**()
java.lang.String	**getSaturday**()
java.lang.String	**getSunday**()
java.lang.String	**getThursday**()
java.lang.String	**getTimeperiod_name**()
java.lang.String	**getTuesday**()
java.lang.String	**getWednesday**()
void	**setAlias**(java.lang.String alias)
void	**setFriday**(java.lang.String friday)
void	**setMonday**(java.lang.String monday)
void	**setSaturday**(java.lang.String saturday)
void	**setSunday**(java.lang.String sunday)
void	**setThursday**(java.lang.String thursday)
void	**setTimeperiod_name**(java.lang.String timeperiod_name)
void	**setTuesday**(java.lang.String tuesday)
void	**setWednesday**(java.lang.String wednesday)

java.lang.String	**toString**()

Constructor Detail

TimeperiodTO

```
public TimeperiodTO()
```

13.8 Paquete nagios.DAO

Class Summary

DAOFactory	clase encargada de la instanciación de una conexión con una base de datos.

13.8.1 Clase DAOFactory

```
java.lang.Object
  └nagios.DAO.DAOFactory
```

Direct Known Subclasses:
OracleDAOFactory

```
public abstract class DAOFactoryextends java.lang.Object
```

clase encargada de la instanciación de una conexión con una base de datos.

Field Summary

static byte	**ORACLE** Constante para la conexión con una base de datos Oracle

Constructor Summary

DAOFactory()	

Method Summary

static DAOFactory	**getDAOFactory**(byte instancia) Instancia un objeto para el acceso a una base de datos del tipo deseado.

Field Detail

ORACLE

```
public static final byte ORACLE
```

Constante para la conexión con una base de datos Oracle

See Also:
Constant Field Values

Constructor Detail

DAOFactory

```
public DAOFactory()
```

Method Detail

getDAOFactory

```
public static DAOFactory getDAOFactory(byte instancia)
```

Instancia un objeto para el acceso a una base de datos del tipo deseado.

Parameters:
`instancia` -
Returns:
Objeto para el acceso a la base de datos.

13.9 Paquete nagios.DAO.interfaces

Interface Summary	
AdmUserDAO	Interfaz con los métodos necesarios para la gestión de los objetos AdmUser, esto es, los usuarios administradores que pueden acceder a la aplicación, en la base de datos Oracle.
CommandDAO	Interfaz con los métodos necesarios para la gestión de los objetos Command en la base de datos Oracle.
ConnectionDAO	Interfaz que define los métodos para tratar una conexión con una base de datos.
ContactDAO	Interfaz con los métodos necesarios para la gestión de los objetos Contact en la base de datos Oracle.
ContactgroupDAO	Interfaz con los métodos necesarios para la gestión de los objetos Contactgroup en la base de datos Oracle.
HostDAO	Interfaz con los métodos necesarios para la gestión de los objetos Host en la base de datos Oracle.
HostDependencyDAO	Interfaz con los métodos necesarios para la gestión de los objetos "Host Dependencies" en la base de datos.
HostEscalationDAO	Interfaz con los métodos necesarios para la gestión de los objetos Host Escalation en la base de datos.
HostgroupDAO	Interfaz con los métodos necesarios para la gestión de los objetos Hostgroup en la base de datos Oracle.
PlatformDAO	Interfaz con los métodos necesarios para la gestión de los objetos Platform en la base de datos Oracle.
ServiceDAO	Interfaz con los métodos necesarios para la gestión de los objetos Services en la base de datos Oracle.
ServicegroupDAO	Interfaz con los métodos necesarios para la gestión de los objetos Servicegroup en la base de datos Oracle.
SrvDependencyDAO	Interfaz con los métodos necesarios para la gestión de los objetos Host Dependency en la base de datos.
SrvEscalationDAO	Interfaz con los métodos necesarios para gestionar los objetos Service Escalation en la base de datos.
TimeperiodDAO	Interfaz con los métodos necesarios para la gestión de los objetos Timeperiod en la base de datos Oracle.

13.9.1 Interfaz AdmUserDAO

```
public interface AdmUserDAO
```

Interfaz con los métodos necesarios para la gestión de los objetos AdmUser, esto es, los usuarios administradores que pueden acceder a la aplicación, en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary

Result[]	**deleteAdmUser**(AdmUserTO[] to) Borra uno o varios AdmUser de la base de datos.
AdmUserTO[]	**findAdmUser**(AdmUserTO to) Busca entre los objetos AdmUser a partir de un filtro de búsqueda, o devuelve todos los AdmUser registrados.
Result[]	**insertAdmUser**(AdmUserTO[] to) Inserta uno o varios AdmUser en la base de datos.
Result[]	**updateUsuariosAdm**(AdmUserTO[] to) Actualiza los atributos de uno o varios AdmUsers ya registrados en la base de datos.

Method Detail

insertAdmUser

```
Result[] insertAdmUser(AdmUserTO[] to)
```

Inserta uno o varios AdmUser en la base de datos.

Parameters:
to - Array de objetos AdmUser que han de ser insertados en la base de datos.
Returns:
Array de resultados de las operaciones de inserción.

---

deleteAdmUser

```
Result[] deleteAdmUser(AdmUserTO[] to)
```

Borra uno o varios AdmUser de la base de datos.

Parameters:
`to` - Array de objetos AdmUser que han de ser borrados o null en caso de que se quiera que todos los usuarios sean borrados.
Returns:
Array de resultados de las operaciones de borrado.

---

findAdmUser

`AdmUserTO[] findAdmUser(AdmUserTO to)`

Busca entre los objetos AdmUser a partir de un filtro de búsqueda, o devuelve todos los AdmUser registrados.

Parameters:
`to` - Objeto AdmUser usado como filtro para la búsqueda o null en caso de que se quiera que se muestren todos los AdmUsers.
Returns:
Array de AdmUsers que se obtienen en la búsqueda.

---

updateUsuariosAdm

`Result[] updateUsuariosAdm(AdmUserTO[] to)`

Actualiza los atributos de uno o varios AdmUsers ya registrados en la base de datos.

Parameters:
`to` - Array de Admusers que se quiere que sean actualizados.
Returns:
Array de resultados de las operaciones de actualización.

13.9.2 Interfaz CommandDAO

`public interface CommandDAO`

Interfaz con los métodos necesarios para la gestión de los objetos Command en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary	
Result[]	**deleteCommand**(CommandTO[] to) Borra un conjunto de objetos Command de la base de datos
CommandTO[]	**findCommand**(CommandTO to) Encuentra un comando o devuelve todos los comandos de la base de datos si se pasa un filtro nulo.
Result[]	**insertCommand**(CommandTO[] to) Inserta un conjunto de objetos Command en la base de datos.
Result[]	**updateCommand**(CommandTO[] to) Actualiza los parámetros de uno o varios Commands insertados con anterioridad en la base de datos.

Method Detail

insertCommand

Result[] **insertCommand**(CommandTO[] to)

Inserta un conjunto de objetos Command en la base de datos.

Parameters:
to - Array de Commands con todos sus parámetros para ser insertados en la base de datos.
Returns:
Array de resultados de cada operación de inserción.

deleteCommand

Result[] **deleteCommand**(CommandTO[] to)

Borra un conjunto de objetos Command de la base de datos

Parameters:
to - Array de objetos que han de ser borrados o null en caso de que se quieran borrar todos.
Returns:
Array de resultados de cada operación de borrado.

findCommand

`CommandTO[] findCommand(CommandTO to)`

Encuentra un comando o devuelve todos los comandos de la base de datos si se pasa un filtro nulo.

Parameters:

`to` - Objeto Command que sirve como filtro de búsqueda en caso de tener incluidos ciertos atributos como el nombre o si el comando es de notificación o de chequeo.

Returns:

Array con los objetos Commands que han sido recogidos de la base de datos en el proceso de búsqueda o null si no se han encontrado Commands.

---

updateCommand

`Result[] updateCommand(CommandTO[] to)`

Actualiza los parámetros de uno o varios Commands insertados con anterioridad en la base de datos.

Parameters:

`to` - Array de objetos Command que han de ser actualizados en la base de datos. Para los objetos Command se podrá actualizar la linea de comando y si el comando es de notificación o de chequeo.

Returns:

Array de resultados de cada operación de actualización.

13.9.3 Interfaz ConnectionDAO

`public interface ConnectionDAO`

Interfaz que define los métodos para tratar una conexión con una base de datos. Los métodos que define se usan para crear, cerrar o comprobar una conexión.

Method Summary	
void	**closeConnection**() Cierra la conexión

void	**createConnection**(java.lang.Object con) Crea la conexión
java.lang.Object	**getConnection**(java.lang.Object con) Obtiene la conexión
boolean	**isConnection**() Comprueba que hay una conexión
java.lang.Object	**reConnect**(java.lang.Object con) Vuelve a reconectar.
void	**setConnection**(java.lang.Object con) Establece la conexión

Method Detail

createConnection

```
void createConnection(java.lang.Object con)
```

Crea la conexión

Parameters:
`con` - Conexión a la base de datos.

---

closeConnection

```
void closeConnection()
```

Cierra la conexión

---

reConnect

```
java.lang.Object reConnect(java.lang.Object con)
```

Vuelve a reconectar.

Parameters:
`con` - Conexión anterior.
Returns:
Nueva conexión a la base de datos.

getConnection

```
java.lang.Object getConnection(java.lang.Object con)
```

Obtiene la conexión

Parameters:
`con` - Conexión
Returns:
Conexión.

setConnection

```
void setConnection(java.lang.Object con)
```

Establece la conexión

Parameters:
`con` - Conexión.

isConnection

```
boolean isConnection()
```

Comprueba que hay una conexión

Returns:
un booleano indicando si hay conexión

13.9.4 Interfaz ContactDAO

```
public interface ContactDAO
```

Interfaz con los métodos necesarios para la gestión de los objetos Contact en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary	
Result[]	**deleteContact**(ContactTO[] to) Borra uno o varios Contacts de la base de datos.
ContactTO[]	**findContact**(ContactTO to) Busca contactos en la base de datos.
ContactTO[]	**findTemplate**(ContactTO to) Busca las posibles plantillas de Contacts existentes en la base de datos.
Result	**insertarTrait**(java.lang.String typename, java.lang.String contactName, Trait Attr) Inserta un atributo del Contact en la tabla de atributos de contactos.
Result[]	**insertContact**(ContactTO[] to) Inserta uno o varios Contacts en la base de datos.
Result[]	**insertTemplate**(ContactTO[] to) Inserta un conjunto de plantillas en la base de datos.
ContactTO	**showContact**(ContactTO to) Muestra un objeto Contact con todos sus parámetros.
Result[]	**updateContact**(ContactTO[] to) Actualiza los parámetros de un conjunto de Contacts.
Result	**updateTrait**(java.lang.String typename, java.lang.String contactName, Trait Attr) Actualiza un atributo del Contact en la tabla de de atributos de contactos.

Method Detail

insertContact

Result[] **insertContact**(ContactTO[] to)

Inserta uno o varios Contacts en la base de datos. Esta inserción se divide en dos partes, primero se insertan los parámetros básicos en una tabla y después el resto de atributos en otra tabla.

Parameters:

to - Array que contiene los objetos Contacts que se quieren insertar en la base de datos.

Returns:

Array de resultados de cada operación de inserción.

deleteContact

`Result[] deleteContact(ContactTO[] to)`

Borra uno o varios Contacts de la base de datos. El borrado del contacto en la tabla de parámetros básicos implica el borrado del resto de atributos ya que están referenciados al contacto.

Parameters:

`to` - Array que contiene los objetos Contacts que se quieren eliminar de la base de datos.

Returns:

Array de resultados de las operaciones de borrado.

findContact

`ContactTO[] findContact(ContactTO to)`

Busca contactos en la base de datos.

Parameters:

`to` - Objeto Contact que sirve de filtro de búsqueda. Se podrán usar ciertos atributos del Contact como parámetros obligatorios del resultado de la búsqueda, como el grupo al que ha de pertenecer o el email.

Returns:

Array con los objetos Contact que han sido recogido en el proceso de búsqueda o null si no se encuentra ninguno. Estos Contacts que se devuelven solo tendrán sus parámetros básicos.

updateContact

`Result[] updateContact(ContactTO[] to)`

Actualiza los parámetros de un conjunto de Contacts.

Parameters:

`to` - Array con los objetos Contact que se quieren actualizar en la base de datos. Se actualizan tanto sus parámetros básicos como los parámetros que se guardan en la tabla de atributos de contactos.

Returns:

Array de resultados de las operaciones de actualización.

---

showContact

`ContactTO` **`showContact`**`(ContactTO to)`

Muestra un objeto Contact con todos sus parámetros.

Parameters:
`to` - Objeto Contact que sirve como filtro de búsqueda. Ha de contener el parámetro "contact_name" el cual es clave de los Contacts y por lo tanto único.
Returns:
Objeto Contact con todos sus parámetros o null si no se ha encontrado nada.

---

findTemplate

`ContactTO[]` **`findTemplate`**`(ContactTO to)`

Busca las posibles plantillas de Contacts existentes en la base de datos.

Parameters:
`to` - Objeto Contact que sirve como filtro de búsqueda.
Returns:
Array de objetos Contacts en los que se devuelven las plantillas con sus parámetros básicos o null en caso de no encontrarse ninguna.

---

insertTemplate

`Result[]` **`insertTemplate`**`(ContactTO[] to)`

Inserta un conjunto de plantillas en la base de datos. Al ser una plantilla no ha de tener todos los parámetros obligatorios que tendría un Contact que si se registra.

Parameters:
`to` - Array de objetos Contact que han de ser insertados en la base de datos.
Returns:
Array de resultados de las operaciones de inserción.

---

insertarTrait

```
Result insertarTrait(java.lang.String typename,
                     java.lang.String contactName,
                     Trait Attr)
```

Inserta un atributo del Contact en la tabla de atributos de contactos.

Parameters:

`typename` - Nombre del atributo.

`contactName` - Nombre del Contact al que pertenece el atributo.

`Attr` - Objeto Trait que se inserta.

Returns:

Objeto Result informando sobre el resultado de la operación de inserción del atributo.

---

updateTrait

```
Result updateTrait(java.lang.String typename,
                   java.lang.String contactName,
                   Trait Attr)
```

Actualiza un atributo del Contact en la tabla de de atributos de contactos.

Parameters:

`typename` - Nombre del atributo.

`contactName` - Nombre del Contact al que pertenece el atributo.

`Attr` - Objeto Trait que se actualiza.

Returns:

Resultado de la operación de actualización.

13.9.5 Interfaz ContactgroupDAO

```
public interface ContactgroupDAO
```

Interfaz con los métodos necesarios para la gestión de los objetos Contactgroup en la base de datos Oracle.

Author:

Agustín Bravo Ortiz

Method Summary	
Result	**addContact**(ContactgroupTO to) Añade un contacto al grupo de contactos insertando el mismo como parámetro del objeto Contactgroup.
Result	**deleteContact**(ContactgroupTO to) Elimina un contacto del grupo de contactos.
Result[]	**deleteContactgroup**(ContactgroupTO[] to) Borra uno o varios Contactgroups de la base de datos.
ContactgroupTO[]	**findContactgroup**(ContactgroupTO to) Busca un Contactgroup en la base de datos en base a su nombre o a sus miembros.
Result[]	**insertContactgroup**(ContactgroupTO[] to) Inserta uno o varios Contactgroups en la base de datos.
ContactgroupTO	**showContactgroup**(ContactgroupTO to) Muestra todos los parámetros básicos de un Contactgroup y sus miembros.
Result[]	**updateContactgroup**(ContactgroupTO[] to) Actualiza los parámetros de uno o varios Contactgroups.

Method Detail

addContact

Result **addContact**(ContactgroupTO to)

Añade un contacto al grupo de contactos insertando el mismo como parámetro del objeto Contactgroup.

Parameters:

to - Objeto Contactgroup que contiene los atributos básicos del Contactgroup para poder identificarlo y además un contacto que se ha de añadir al grupo ya creado.

Returns:

Resultado de la inserción del contacto en el grupo.

---

deleteContact

Result **deleteContact**(ContactgroupTO to)

Elimina un contacto del grupo de contactos.

Parameters:
`to` - Objeto Contactgroup que contiene los atributos básicos del contactgroup para poder identificarlo y además un contacto que es el que eliminamos del grupo.
Returns:
Resultado de la operación de eliminar el contacto del grupo.

---

insertContactgroup

`Result[] insertContactgroup(ContactgroupTO[] to)`

Inserta uno o varios Contactgroups en la base de datos. La inserción consta primero de la inserción de los parámetros que identifican al contactgroup, y segundo de los miembros del mismo.

Parameters:
`to` - Objetos Contactgroups que se han de insertar en la base de datos.
Returns:
Resultados de las operaciones de inserción.

---

deleteContactgroup

`Result[] deleteContactgroup(ContactgroupTO[] to)`

Borra uno o varios Contactgroups de la base de datos. Debido a la configuración usada en las tablas al borrar el Contactgroup se borrarán automáticamente las entradas del mismo en la tabla de miembros.

Parameters:
`to` - Objetos Contactgroup que se han de borrar o null en caso de que se quieran borrar todos los Contactgroups.
Returns:
Array de resultados de todas las operaciones de borrado.

---

findContactgroup

`ContactgroupTO[] findContactgroup(ContactgroupTO to)`

Busca un Contactgroup en la base de datos en base a su nombre o a sus miembros.

Parameters:
`to` - Objeto Contactgroup que sirve como filtro de búsqueda. Podrá ser null para el caso en que se quieran buscar todos los Contactgroups.
Returns:
Array de Contactgroups de los objetos que se han encontrado en la base de datos o todos los Contactgroups en caso de que el parámetro de entrada tuviese valor nulo. Se devuelven solo los parámetros básicos del Contactgroup y no sus miembros.

---

showContactgroup

`ContactgroupTO` **`showContactgroup`**`(ContactgroupTO to)`

Muestra todos los parámetros básicos de un Contactgroup y sus miembros.

Parameters:
`to` - Objeto Contactgroup que sirve como filtro de búsqueda. Ha de tener el parámetro contactgoup_name que identifica al Contactgroup de forma unívoca.
Returns:
Objeto Contactgroup con todos sus parámetros y miembros.

---

updateContactgroup

`Result[]` **`updateContactgroup`**`(ContactgroupTO[] to)`

Actualiza los parámetros de uno o varios Contactgroups.

Parameters:
`to` - Objeto Contactgroup que se actualiza.
Returns:
Array de resultados de cada operación de actualización.

13.9.6 Interfaz HostDAO

`public interface` **`HostDAO`**

Interfaz con los métodos necesarios para la gestión de los objetos Host en la base de datos Oracle.

Author:

Agustín Bravo Ortiz

Method Summary	
Result[]	**deleteHost**(HostTO[] to) Borra uno o varios Hosts de la base de datos.
Result	**deleteTrait**(java.lang.String typename, java.lang.String hostName, Trait Attr) Borra un atributo de un Host de la tabla de atributos.
HostTO[]	**findHost**(HostTO to) Encuentra un Host en base a un filtro o los muestra todos.
HostTO[]	**findTemplate**(HostTO to) Busca plantillas de Hosts en la base de datos, las cuales vendrán caracterizadas porque su nombre empieza por "plnt_".
Result[]	**insertHost**(HostTO[] to) Inserta uno o varios Hosts en la base de datos.
Result	**insertTrait**(java.lang.String typename, java.lang.String hostName, Trait Attr) Inserta un atributo de un Host en la tabla de atributos.
HostTO	**showHost**(HostTO to) Muestra todos los parámetros de un Host.
Result[]	**updateHost**(HostTO[] to) Actualiza los parámetros de uno o varios Hosts.
Result	**updateTrait**(java.lang.String typename, java.lang.String hostName, Trait Attr) Actualiza un atributo de un Host en la tabla de atributos.

Method Detail

insertHost

Result[] **insertHost**(HostTO[] to)

Inserta uno o varios Hosts en la base de datos. Estos han de tener ciertos atributos obligatoriamente, vienen indicados por la configuración de Nagios.

Parameters:

`to` - Array de Hosts que han de ser insertados en la base de datos.

Returns:

Array de resultados de todas la operaciones de inserción.

---

deleteHost

`Result[] deleteHost(HostTO[] to)`

Borra uno o varios Hosts de la base de datos. Debido a la configuración de las tablas el borrado de un Host conllevará el borrado de sus atributos de la tabla de atributos.

Parameters:
`to` - Array de Hosts que se quiere que sean borrados o null en caso de que se quieran borrar todos los Hosts registrados.
Returns:
Array de resultados de todas las operaciones de borrado.

---

findHost

`HostTO[] findHost(HostTO to)`

Encuentra un Host en base a un filtro o los muestra todos.

Parameters:
`to` - Objeto Host que sirve como filtro de búsqueda.
Returns:
Array de Hosts que concuerdan con el filtro pasado para la búsqueda o todos los Hosts que están registrados en caso de haber recibido null como parámetro de entrada. Los objetos Hosts que se devuelven sólo tendrán los parámetros básicos.

---

updateHost

`Result[] updateHost(HostTO[] to)`

Actualiza los parámetros de uno o varios Hosts.

Parameters:
`to` - Array de Hosts a los que se han de actualizar los parámetros.
Returns:
Array de resultados de las operaciones de actualización.

---

showHost

```
HostTO showHost(HostTO to)
```

Muestra todos los parámetros de un Host.

Parameters:

`to` - Objeto Host que se usa como filtro de búsqueda. Ha de contener al menos el parámetro host_name que identifica al Host de forma unívoca.

Returns:

Objeto Host con todos los parámetros con los que está registrado en la base de datos.

---

findTemplate

```
HostTO[] findTemplate(HostTO to)
```

Busca plantillas de Hosts en la base de datos, las cuales vendrán caracterizadas porque su nombre empieza por "plnt_".

Parameters:

`to` - Filtro de búsqueda que usualmente se llamará con valor null para que devuelva todas las plantillas.

Returns:

Array de Hosts con sus parámetros básicos.

---

updateTrait

```
Result updateTrait(java.lang.String typename,
                   java.lang.String hostName,
                   Trait Attr)
```

Actualiza un atributo de un Host en la tabla de atributos.

Parameters:

`typename` - Nombre del atributo que se actualiza.

`hostName` - Nombre del host al que se refiere el atributo.

`Attr` - Atributo que se actualiza en la tabla.

Returns:

Resultado de la operación de actualización.

---

insertTrait

```
Result insertTrait(java.lang.String typename,
                   java.lang.String hostName,
                   Trait Attr)
```

Inserta un atributo de un Host en la tabla de atributos.

Parameters:

`typename` - Nombre del atributo que se inserta.
`hostName` - Nombre del Host al que se refiere el atributo.
`Attr` - Atributo que se inserta en la tabla.

Returns:

Resultado de la operación de inserción.

---

deleteTrait

```
Result deleteTrait(java.lang.String typename,
                   java.lang.String hostName,
                   Trait Attr)
```

Borra un atributo de un Host de la tabla de atributos.

Parameters:

`typename` - Nombre del atributo que se borra.
`hostName` - Nombre del Host al que se refiere el atributo.
`Attr` - Atributo que se borra.

Returns:

Resultado de la operación de borrado.

13.9.7 Interfaz HostgroupDAO

```
public interface HostgroupDAO
```

Interfaz con los métodos necesarios para la gestión de los objetos Hostgroup en la base de datos Oracle.

Author:

Agustín Bravo Ortiz

Method Summary	
Result	**addHost**(HostgroupTO to) Añade un Host a la lista de miembros del Hostgroup.
Result	**deleteHost**(HostgroupTO to) Elimina un Host del grupo de miembros del Hostgroup.
Result[]	**deleteHostgroup**(HostgroupTO[] to) Elimina uno o varios Hostgroup de la base de datos.
HostgroupTO[]	**findHostgroup**(HostgroupTO to) Busca Hostgroups en la base de datos usando un filtro de búsqueda, o muestra todos los Hostgroups.
Result[]	**insertHostgroup**(HostgroupTO[] to) Inserta uno o varios Hostgroups en la base de datos.
HostgroupTO	**showHostgroup**(HostgroupTO to) Muestra un Hostgroup con todos sus parámetros.
Result[]	**updateHostgroup**(HostgroupTO[] to) Actualiza los parámetros de un Hostgroup.

Method Detail

addHost

Result **addHost**(HostgroupTO to)

Añade un Host a la lista de miembros del Hostgroup.

Parameters:

to - Objeto Hostgroup que contiene los parámetros básicos que identifican el Hostgroup y un Host como miembro que es el que se añade al grupo de miemrbros.

Returns:

Resultado de la operación de añadir el Host al grupo.

---

deleteHost

Result **deleteHost**(HostgroupTO to)

Elimina un Host del grupo de miembros del Hostgroup.

Parameters:

`to` - Objeto Hostgroup que contiene los parámetros básicos que identifican el Hostgroup y un Host como miembro que el el que se ha de eliminar del grupo de miembros.

Returns:

Resultado de la operación de eliminar el Host del grupo.

---

insertHostgroup

`Result[] insertHostgroup(HostgroupTO[] to)`

Inserta uno o varios Hostgroups en la base de datos. La inserción consta primero de la inserción de los parámetros básicos del Hostgroup y segundo la inserción de los miembros que los forman.

Parameters:

`to` - Array de Hostgroups que se han de insertar en la base de datos.

Returns:

Array de resultados de las operaciones de inserción.

---

deleteHostgroup

`Result[] deleteHostgroup(HostgroupTO[] to)`

Elimina uno o varios Hostgroup de la base de datos. Por la configuración de las tablas, al borrar un Hostgroup se eliminarán automáticamente las indicaciones del grupo de miembros que lo conforman.

Parameters:

`to` - Array de Hostgroups que han de ser borrados o null en caso de que haya que borrar todos los Hostgroups.

Returns:

Array de resultados de las operaciones de borrado.

---

findHostgroup

`HostgroupTO[] findHostgroup(HostgroupTO to)`

Busca Hostgroups en la base de datos usando un filtro de búsqueda, o muestra todos los Hostgroups.

Parameters:
`to` - Objeto Hostgroup que sirve como filtro de búsqueda.
Returns:
Array de Hostgroups que coinciden con el filtro de búsqueda o todos los Hostgroups en caso de que se haya insertado null como filtro. Los Hostgroups que se devuelven solo tendrán sus parámetros básicos y no los miembros que los componen.

---

updateHostgroup

`Result[] updateHostgroup(HostgroupTO[] to)`

Actualiza los parámetros de un Hostgroup.

Parameters:
`to` - Objeto Hostgroup que hay que actualizar con sus parámetros ya actualizados.
Returns:
Array de resultados de las operaciones de actualización.

---

showHostgroup

`HostgroupTO showHostgroup(HostgroupTO to)`

Muestra un Hostgroup con todos sus parámetros.

Parameters:
`to` - Objeto Hostgroup que nos sirve como filtro de búsqueda. Ha de contener el atributo hostgroup_name que nos identifica el Hostgroup de forma unívoca.
Returns:
Objeto Hostgroup con todos sus parámetros y miembros que lo componen.

13.9.8 Interfaz PlatformDAO

`public interface PlatformDAO`

Interfaz con los métodos necesarios para la gestión de los objetos Platform en la base de datos Oracle.

Author:

Agustín Bravo Ortiz

Method Summary	
Result[]	**deletePlatform**(PlatformTO[] to) Borra uno o varios Platform de la base de datos.
PlatformTO[]	**findPlatform**(PlatformTO to) Busca Platforms en la base de datos usando un filtro de búsqueda, o muestra todos los Platform existentes.
Result[]	**insertPlatform**(PlatformTO[] to) Inserta uno o varios Platform en la base de datos.
Result[]	**updatePlatform**(PlatformTO[] to) Actualiza los parámetros de uno o varios Platforms en la base de datos.

Method Detail

insertPlatform

Result[] **insertPlatform**(PlatformTO[] to)

Inserta uno o varios Platform en la base de datos.

Parameters:
to - Array de objetos Platform que han de ser insertados en la base de datos.
Returns:
Array de resultados de las operaciones de inserción.

deletePlatform

Result[] **deletePlatform**(PlatformTO[] to)

Borra uno o varios Platform de la base de datos.

Parameters:
to - Array de objetos Platform que identifican a los que hay que borrar o null en caso de que se quieran borrar todos los Platform de la base de datos.
Returns:
Array de resultados de las operaciones de borrado.

findPlatform

`PlatformTO[] findPlatform(PlatformTO to)`

Busca Platforms en la base de datos usando un filtro de búsqueda, o muestra todos los Platform existentes.

Parameters:
`to` - Objeto Platform que sirve de filtro o null en caso de que se quiera que se muestren todos los Platform de la base de datos.
Returns:
Array de Platforms encontrados.

---

updatePlatform

`Result[] updatePlatform(PlatformTO[] to)`

Actualiza los parámetros de uno o varios Platforms en la base de datos.

Parameters:
`to` - Array de Platforms que han de ser actualizados en la base de datos
Returns:
Array de resultados de las operaciones de actualización.

13.9.9 Interfaz ServiceDAO

`public interface ServiceDAO`

Interfaz con los métodos necesarios para la gestión de los objetos Services en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary	
`Result[]`	**`deleteService`**`(ServiceTO[] to)` Borra uno o varios Services de la base de datos.
`Result`	**`deleteTrait`**`(java.lang.String typename,`

	java.lang.String id_srv, Trait Attr) Borra un atributo de un Service de la tabla de atributos.
ServiceTO[]	**findService**(ServiceTO to) Busca Services en la base de datos usando un filtro de búsqueda, o muestra toddos los Services existentes.
ServiceTO[]	**findTemplate**(ServiceTO to) Busca plantillas de Service en la base de datos, las cuales vendrán caracterizadas porque su atributo service_description comienza por "plnt_".
Result[]	**insertService**(ServiceTO[] to) Inserta uno o varios Services en la base de datos.
Result	**insertTrait**(java.lang.String typename, java.lang.String id_srv, Trait Attr) Inserta un atributo de un Service en la tabla de atributos.
ServiceTO	**showService**(ServiceTO to) Muestra todos los parámetros de un Service.
Result[]	**updateService**(ServiceTO[] to) Actualiza los parámetros de uno o varios Services.
Result	**updateTrait**(java.lang.String typename, java.lang.String id_srv, Trait Attr) Actualiza un atributo de un Service en la tabla de atributos.

Method Detail

insertService

Result[] **insertService**(ServiceTO[] to)

Inserta uno o varios Services en la base de datos.

Parameters:
to - Array de Services que han de ser insertados en la base de datos.
Returns:
Array de resultados de las operaciones de inserción.

---

deleteService

Result[] **deleteService**(ServiceTO[] to)

Borra uno o varios Services de la base de datos.

Parameters:
`to` - Array de Services que han de ser borrados o null en caso de que se quiera que todos los Services sean borrados.
Returns:
Array de resultados de las operaciones de borrado.

---

findService

`ServiceTO[] findService(ServiceTO to)`

Busca Services en la base de datos usando un filtro de búsqueda, o muestra todos los Services existentes.

Parameters:
`to` - Objeto Service que se usa como filtro de búsqueda o null en caso de que se quiere que se muestren todos los Services registrados.
Returns:
Array de Services encontrados. Solo tendrán los atributos básicos.

---

updateService

`Result[] updateService(ServiceTO[] to)`

Actualiza los parámetros de uno o varios Services.

Parameters:
`to` - Array de Services a los que hay que actualizar los parámetros.
Returns:
Array de resultados d las operaciones de actualización.

---

findTemplate

`ServiceTO[] findTemplate(ServiceTO to)`

Busca plantillas de Service en la base de datos, las cuales vendrán caracterizadas porque su atributo service_description comienza por "plnt_".

Parameters:
`to` - Filtro de búsqueda que usualmente será null para que el método devuelva todas las plantillas existentes.

Returns:
Array de Services con sus parámetros básicos.

---

showService

```
ServiceTO showService(ServiceTO to)
```

Muestra todos los parámetros de un Service.

Parameters:
`to` - Objeto Service que se usa como filtro de búsqueda. Ha de tener al menos el identificador del servicio que identifica de forma unívoca dicho servicio.
Returns:
Objeto Service con todos sus atributos con los que está registrado en la base de datos.

---

insertTrait

```
Result insertTrait(java.lang.String typename,
                   java.lang.String id_srv,
                   Trait Attr)
```

Inserta un atributo de un Service en la tabla de atributos.

Parameters:
`typename` - Nombre del atributo.
`id_srv` - Identificador del Service al que corresponde el atributo.
`Attr` - Atributo que se inserta.
Returns:
Resultado de la operación de inserción.

---

updateTrait

```
Result updateTrait(java.lang.String typename,
                   java.lang.String id_srv,
                   Trait Attr)
```

Actualiza un atributo de un Service en la tabla de atributos.

Parameters:
`typename` - Nombre del atributo.

`id_srv` - Identificador del Service al que corresponde el atributo.
`Attr` - Atributo que se actualiza.

Returns:
Resultado de la operación de actualización.

---

deleteTrait

```
Result deleteTrait(java.lang.String typename,
                   java.lang.String id_srv,
                   Trait Attr)
```

Borra un atributo de un Service de la tabla de atributos.

Parameters:
`typename` - Nombre del atributo.
`id_srv` - Identificador del Service al que corresponde el atributo.
`Attr` - Atributo que se borra.

Returns:
Resultado de la operación de borrado.

13.9.10 Interfaz ServicegroupDAO

```
public interface ServicegroupDAO
```

Interfaz con los métodos necesarios para la gestión de los objetos Servicegroup en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary	
Result	**addService**(ServicegroupTO to) Añade un servicio al grupo de miembros del Servicegroup.
Result	**deleteService**(ServicegroupTO to) Elimina un servicio del grupo de miembros del Servicegroup.
Result[]	**deleteServicegroup**(ServicegroupTO[] to) Borra uno o varios Servicegroups de la base de datos.

ServicegroupTO[]	**findServicegroup**(ServicegroupTO to) Busca Servicegroups en la base de datos usando un filtro de búsqueda, o muestra todos los Servicegroups registrados.
Result[]	**insertServicegroup**(ServicegroupTO[] to) Inserta uno o varios Servicegroups en la base de datos.
ServicegroupTO	**showServicegroup**(ServicegroupTO to) Muestra un Servicegroup con todos sus parámetros.
Result[]	**updateServicegroup**(ServicegroupTO[] to) Actualiza los parámetros y miembros de uno o varios Servicegroups en la base de datos.

Method Detail

addService

Result **addService**(ServicegroupTO to)

Añade un servicio al grupo de miembros del Servicegroup.

Parameters:
to - Objeto Servicegroup que contiene los atributos básicos que definen al Servicegroup y un Service como miembro que es el que se añade al Servicegroup.
Returns:
Resultado de la operación de insertar el Service en el grupo.

---

deleteService

Result **deleteService**(ServicegroupTO to)

Elimina un servicio del grupo de miembros del Servicegroup.

Parameters:
to - Objeto Servicegropu que contiene los atributos básicos que definen al Servicegroup y un Service como miembro que es que se elimina del Servicegroup.
Returns:
Resultado de la operación de eliminar el Service del grupo.

---

insertServicegroup

`Result[] insertServicegroup(ServicegroupTO[] to)`

Inserta uno o varios Servicegroups en la base de datos. La inserción constan primero de la inserción de los parámetros básicos del Servicegroup y segundo la inserción de los miembros que componen el Servicegroup.

Parameters:
`to` - Array de Servicegroup que han de ser insertados.
Returns:
Array de resultados de las operaciones de inserción.

---

deleteServicegroup

`Result[] deleteServicegroup(ServicegroupTO[] to)`

Borra uno o varios Servicegroups de la base de datos.

Parameters:
`to` - Array de Servicegroups que han de ser borrados o null en caso de que se quiera que todos los Servicegroups sean borrados.
Returns:
Array de resultados de las operaciones de borrado.

---

findServicegroup

`ServicegroupTO[] findServicegroup(ServicegroupTO to)`

Busca Servicegroups en la base de datos usando un filtro de búsqueda, o muestra todos los Servicegroups registrados.

Parameters:
`to` - Objeto Servicegroup que sirve como filtro de búsqueda o null en caso de que se quiera que devuelva todos los Servicegroups registrados.
Returns:
Array de Servicegroup que se obtienen de la búsqueda.

---

updateServicegroup

`Result[] updateServicegroup(ServicegroupTO[] to)`

Actualiza los parámetros y miembros de uno o varios Servicegroups en la base de datos.

Parameters:
`to` - Array de Servicegroups que han de ser actualizados.
Returns:
Array de resultados de las operaciones de actualización.

---

showServicegroup

`ServicegroupTO` **`showServicegroup`**`(ServicegroupTO to)`

Muestra un Servicegroup con todos sus parámetros.

Parameters:
`to` - Objeto Servicegroup que sirve como filtro de búsqueda y que ha de contener al menos el atributo servicegroup_name que identifica al Servicegroup unívocamente en la base de datos.
Returns:
Objeto Servicegroup con todos sus parámetros y miembros.

13.9.11 Interfaz TimeperiodDAO

`public interface` **`TimeperiodDAO`**

Interfaz con los métodos necesarios para la gestión de los objetos Timeperiod en la base de datos Oracle.

Author:
Agustín Bravo Ortiz

Method Summary	
`Result[]`	**`deleteTimeperiod`**`(TimeperiodTO[] to)` Borra uno o varios Timeperiods de la base de datos.
`TimeperiodTO[]`	**`findTimeperiod`**`(TimeperiodTO to)` Busca un Timeperiod a partir de un filtro de búsqueda, o muestra todos los existentes.
`Result[]`	**`insertTimeperiod`**`(TimeperiodTO[] to)` Inserta uno o varios Timeperiods en la base de datos.

Result[]	**updateTimeperiod**(TimeperiodTO[] to) Actualiza los atributos de uno o varios Timeperiods en la base de datos.

Method Detail

insertTimeperiod

Result[] **insertTimeperiod**(TimeperiodTO[] to)

Inserta uno o varios Timeperiods en la base de datos.

Parameters:
to - Array de Timeperiods que se insertan en la base de datos.
Returns:
Array de resultados de las operaciones de inserción.

---

deleteTimeperiod

Result[] **deleteTimeperiod**(TimeperiodTO[] to)

Borra uno o varios Timeperiods de la base de datos.

Parameters:
to - Array de Timeperiods que han de ser borrados o null en caso de que se quiera que se borren todos los Timeperiods.
Returns:
Array de resultados de las operaciones de borrado.

---

findTimeperiod

TimeperiodTO[] **findTimeperiod**(TimeperiodTO to)

Busca un Timeperiod a partir de un filtro de búsqueda, o muestra todos los existentes.

Parameters:
to - Objeto Timeperiod que nos sirve como filtro de búsqueda en caso de no ser nulo y de que contenga el atributo timeperiod_name. O también puede ser null y que la función devuelva todos los Timeperiods registrados en la base de datos.
Returns:

Array de Timeperiods que se obtienen en la búsqueda.

---

updateTimeperiod

`Result[] updateTimeperiod(TimeperiodTO[] to)`

Actualiza los atributos de uno o varios Timeperiods en la base de datos.

Parameters:
`to` - Array de objetos Timeperiod que han de ser actualizados.
Returns:
Array de resultados de las operaciones de actualización.

13.10 Paquete nagios.DAO.oracle

Class Summary	
OracleAdmUserDAO	Clase encargada de la gestión de los objetos AdmUsers en la base de datos.
OracleCommandDAO	Clase encargada de la gestión de los objetos Command en la base de datos.
OracleContactDAO	Clase encargada de la gestión de los objetos Command en la base de datos.
OracleContactgroupDAO	Clase encargada de la gestión de los objetos Command en la base de datos.
OracleDAOFactory	Clase encargada de la gestión de conexiones a la base de datos de la aplicación.
OracleHostDAO	Clase encargada de la gestión de los objetos Host en la base de datos.
OracleHostDependencyDAO	Clase encargada de la gestión de objetos "Host Dependency" en la base de datos.
OracleHostEscalationDAO	Clase encargada de la gestión de los objetos Host Escalation en la base de datos.
OracleHostgroupDAO	Clase encargada de la gestión de los objetos Hostgroup en la base de datos.

OraclePlatformDAO	Clase encargada de la gestión de los objetos Platform en la base de datos.
OracleServiceDAO	Clase encargada de la gestión de los objetos Services en la base de datos.
OracleServicegroupDAO	Clase encargada de la gestión de los objetos Servicegroups en la base de datos.
OracleSrvDependencyDAO	Clase encargada de la gestión de objetos Service Dependency en la base de datos.
OracleSrvEscalationDAO	Clase encargada de la gestión de objetos Service Escalations en la base de datos.
OracleTimeperiodDAO	Clase encargada de la gestión de los objetos Timeperiods en la base de datos.

13.10.1 Clase OracleAdmUserDAO

```
java.lang.Object
  └nagios.DAO.oracle.OracleAdmUserDAO
```

All Implemented Interfaces:
AdmUserDAO

```
public class OracleAdmUserDAOextends java.lang.Objectimplements AdmUserDAO
```

Clase encargada de la gestión de los objetos AdmUsers en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleAdmUserDAO() Constructor sin parámetros.	

Method Summary

Result[]	**deleteAdmUser**(AdmUserTO[] to) Elimina los usuarios administradores existentes en la base de datos, en

	base a los filtros que se le pasen como parámetro.
AdmUserTO[]	**findAdmUser**(AdmUserTO to) Devuelve los usuarios administradores que existen en la base de datos, en base a los filtros que se le pasen como parámetros.
Result[]	**insertAdmUser**(AdmUserTO[] to) Inserta en la base de datos los datos de los usuarios administradores pasados como parámetros
Result[]	**updateUsuariosAdm**(AdmUserTO[] to) Actualiza los datos de los usuarios administradores pasados como parámetros

Constructor Detail

OracleAdmUserDAO

```
public OracleAdmUserDAO()
```

Constructor sin parámetros.

13.10.2 Clase OracleCommandDAO

```
java.lang.Object
  └nagios.DAO.oracle.OracleCommandDAO
```

All Implemented Interfaces:
CommandDAO

```
public class OracleCommandDAOextends java.lang.Objectimplements CommandDAO
```

Clase encargada de la gestión de los objetos Command en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleCommandDAO() Constructor sin parámetros	

Method Summary	
Result[]	**deleteCommand**(CommandTO[] to) Borra un conjunto de objetos Command de la base de datos
CommandTO[]	**findCommand**(CommandTO to) Encuentra un comando o devuelve todos los comandos de la base de datos si se pasa un filtro nulo.
Result[]	**insertCommand**(CommandTO[] to) Inserta un conjunto de objetcos Command en la base de datos.
Result[]	**updateCommand**(CommandTO[] to) Actualiza los parámetros de uno o varios Commands insertados con anterioridad en la base de datos.

Constructor Detail

OracleCommandDAO

```
public OracleCommandDAO()
```

Constructor sin parámetros

13.10.3 Clase OracleContactDAO

```
java.lang.Object
  └nagios.DAO.oracle.OracleContactDAO
```

All Implemented Interfaces:
ContactDAO

```
public class OracleContactDAOextends java.lang.Objectimplements ContactDAO
```

Clase encargada de la gestión de los objetos Command en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleContactDAO() Constructor sin parámetros.	

Method Summary

Result[]	**deleteContact**(ContactTO[] to) Borra uno o varios Contacts de la base de datos.
ContactTO[]	**findContact**(ContactTO to) Busca contactos en la base de datos.
ContactTO[]	**findTemplate**(ContactTO to) Busca las posibles plantillas de Contacts existentes en la base de datos.
Result	**insertarTrait**(java.lang.String typename, java.lang.String id_contact, Trait Attr) Inserta un atributo del Contact en la tabla de atributos de contactos.
Result[]	**insertContact**(ContactTO[] to) Inserta uno o varios Contacts en la base de datos.
Result[]	**insertTemplate**(ContactTO[] to) Inserta un conjunto de plantillas en la base de datos.
ContactTO	**showContact**(ContactTO to) Muestra un objeto Contact con todos sus parámetros.
Result[]	**updateContact**(ContactTO[] to) Actualiza los parámetros de un conjunto de Contacts.
Result	**updateTrait**(java.lang.String typename, java.lang.String id_contact, Trait Attr) Actualiza un atributo del Contact en la tabla de de atributos de contactos.

Constructor Detail

OracleContactDAO

```
public OracleContactDAO()
```

Constructor sin parámetros.

13.10.4 Clase OracleContactgroupDAO

```
java.lang.Object
  └ nagios.DAO.oracle.OracleContactgroupDAO
```

All Implemented Interfaces:
ContactgroupDAO

```
public class OracleContactgroupDAOextends java.lang.Objectimplements
ContactgroupDAO
```

Clase encargada de la gestión de los objetos Command en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleContactgroupDAO() Constructor sin parámetros.	

Method Summary

Result	**addContact**(ContactgroupTO to) Añade un contacto al grupo de contactos insertándo el mismo como parámetro del objeto Contactgroup.
Result	**deleteContact**(ContactgroupTO to) Elimina un contacto del grupo de contactos.
Result[]	**deleteContactgroup**(ContactgroupTO[] to) Borra uno o varios Contactgroups de la base de datos.
ContactgroupTO[]	**findContactgroup**(ContactgroupTO to) Busca un Contactgroup en la base de datos en base a su nombre o a sus miembros.
Result[]	**insertContactgroup**(ContactgroupTO[] to) Inserta uno o varios Contactgroups en la base de datos.
ContactgroupTO	**showContactgroup**(ContactgroupTO to) Muestra todos los parámetros básicos de un Contactgroup y sus miembros.
Result[]	**updateContactgroup**(ContactgroupTO[] to)

	Actualiza los parámetros de uno o varios Contactgroups.

Constructor Detail

OracleContactgroupDAO

```
public OracleContactgroupDAO()
```

Constructor sin parámetros.

13.10.5 Clase OracleDAOFactory

```
java.lang.Object
  └ nagios.DAO.DAOFactory
      └ nagios.DAO.oracle.OracleDAOFactory
```

All Implemented Interfaces:
ConnectionDAO

```
public class OracleDAOFactory extends DAOFactory implements ConnectionDAO
```

Clase encargada de la gestión de conexiones a la base de datos de la aplicación. En esta clase se crean las conexiones y los objetos de acceso a la base de datos para los distintos objetos de configuración.

Field Summary

Fields inherited from class nagios.DAO.DAOFactory
ORACLE

Method Summary

java.lang.Object	**clone**()
void	**closeConnection**() Cierra la conexion
void	**createConnection**(java.lang.Object ds)

	Crea la conexion
AdmUserDAO	**getAdmUserDAO**() Crea una conexión a la base de datos si no existe aún y un objeto para el acceso a los usuarios de administración de la base de datos.
CommandDAO	**getCommandDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Check Commands de la configuración de Nagios almacenados en la base de datos.
java.lang.Object	**getConnection**(java.lang.Object ds) Obtiene la conexión
ContactDAO	**getContactDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Contacts de la configuración de Nagios almacenados en la base de datos.
ContactgroupDAO	**getContactgroupDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Contactgroups de la configuración de Nagios almacenados en la base de datos.
HostDAO	**getHostDAO**() Crea una conexión a la base de datos si no existe aún y un objeto para el acceso a los Hosts de la configuración de Nagios almacenados en la base de datos.
HostDependencyDAO	**getHostDependencyDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los objetos Host Dependency, usados para configurar dependencies entre Hosts y que on se envíen notificaciones espúreas ante la caida de un Host del que dependen otros.
HostEscalationDAO	**getHostEscalationDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los objetos Service Dependency, usados para configurar dependencias entre servicios y que no se manden notificaciones espúreas ante la caida de un servicio del que dependen otros.
HostgroupDAO	**getHostgroupDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Hostgroups de la configuración de Nagios almacenados en la base de datos.
static OracleDAOFactory	**getInstance**() Genera una instancia de la clase OracleDAOFactory

	para poder usarla posteriormente en la creación de una conexión y su gestión.
PlatformDAO	**getPlatformDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Platforms, los cuales son útiles para agrupar objetos de configuración de Nagios, aunque no formarán parte de la configuración.
ServiceDAO	**getServiceDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Services de la configuración de Nagios almacenados en la base de datos.
ServicegroupDAO	**getServicegroupDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Servicegroups de la configuración de Nagios almacenados en la base de datos.
SrvDependencyDAO	**getSrvDependencyDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los objetos Service Dependency, usados para configurar dependencias entre servicios y que no se manden notificaciones espúreas ante la caída de un servicio del que dependen otros.
SrvEscalationDAO	**getSrvEscalationDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los objetos Service Escalation, usados para escalar las notificaciones a los usuarios del sistema.
TimeperiodDAO	**getTimeperiodDAO**() Crea una conexión a la base de datos si no existe y un objeto para el acceso a los Timeperiods de la configuración de Nagios almacenados en la base de datos.
boolean	**isConnection**() Comprueba que hay una conexión
java.lang.Object	**reConnect**(java.lang.Object ds) Vuelve a reconectar.
void	**setConnection**(java.lang.Object conn) Establece la conexión

13.10.6 Clase OracleHostDAO

```
java.lang.Object
  └nagios.DAO.oracle.OracleHostDAO
```

All Implemented Interfaces:
HostDAO

```
public class OracleHostDAOextends java.lang.Objectimplements HostDAO
```

Clase encargada de la gestión de los objetos Host en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos, además de gestionar por separado sus atributos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleHostDAO() Constructor sin parámetros	

Method Summary

Result[]	**deleteHost**(HostTO[] to) Borra uno o varios Hosts de la base de datos.
Result	**deleteTrait**(java.lang.String typename, java.lang.String id_host, Trait Attr) Borra un atributo de un Host de la tabla de atributos.
HostTO[]	**findHost**(HostTO to) Encuentra un Host en base a un filtro o los muestra todos.
HostTO[]	**findTemplate**(HostTO to) Busca plantillas de Hosts en la base de datos, las cuales vendrán caracterizadas porque su nombre empieza por "plnt_".
Result[]	**insertHost**(HostTO[] to) Inserta uno o varios Hosts en la base de datos.
Result	**insertTrait**(java.lang.String typename, java.lang.String id_host, Trait Attr) Inserta un atributo de un Host en la tabla de atributos.
HostTO	**showHost**(HostTO to) Este método Obtiene todos los parámetros de un host de la base de datos
Result[]	**updateHost**(HostTO[] to) Actualiza los parámetros de uno o varios Hosts.
Result	**updateTrait**(java.lang.String typename,

	java.lang.String id_host, Trait Attr) Actualiza un atributo de un Host en la tabla de atributos.

Constructor Detail

OracleHostDAO

public **OracleHostDAO**()

Constructor sin parámetros

13.10.7 Clase OracleHostgroupDAO

java.lang.Object
└ **nagios.DAO.oracle.OracleHostgroupDAO**

All Implemented Interfaces:
HostgroupDAO

---

public class **OracleHostgroupDAO**extends java.lang.Objectimplements HostgroupDAO

Clase encargada de la gestión de los objetos Hostgroup en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos. También se permite la gestión de sus miembros, adición y sustracción.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleHostgroupDAO() Constructor sin parámetros.	

Method Summary

Result	**addHost**(HostgroupTO to) Añade un Host a la lista de miembros del Hostgroup.
Result	**deleteHost**(HostgroupTO to)

	Elimina un Host del grupo de miembros del Hostgroup.
Result[]	**deleteHostgroup**(HostgroupTO[] to) Elimina uno o varios Hostgroup de la base de datos.
HostgroupTO[]	**findHostgroup**(HostgroupTO to) Busca Hostgroups en la base de datos usando un filtro de búsqueda, o muestra todos los Hostgroups.
Result[]	**insertHostgroup**(HostgroupTO[] to) Inserta uno o varios Hostgroups en la base de datos.
HostgroupTO	**showHostgroup**(HostgroupTO to) Muestra un Hostgroup con todos sus parámetros.
Result[]	**updateHostgroup**(HostgroupTO[] to) Actualiza los parámetros de un Hostgroup.

Constructor Detail

OracleHostgroupDAO

```
public OracleHostgroupDAO()
```

Constructor sin parámetros.

13.10.8 Clase OraclePlatformDAO

```
java.lang.Object
  └nagios.DAO.oracle.OraclePlatformDAO
```

All Implemented Interfaces:
PlatformDAO

```
public class OraclePlatformDAOextends java.lang.Objectimplements
PlatformDAO
```

Clase encargada de la gestión de los objetos Platform en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary	
OraclePlatformDAO() Constructor sin parámetros.	

Method Summary	
Result[]	**deletePlatform**(PlatformTO[] to) Borra uno o varios Platform de la base de datos.
PlatformTO[]	**findPlatform**(PlatformTO to) Busca Platforms en la base de datos usando un filtro de búsqueda, o muestra todos los Platform existentes.
Result[]	**insertPlatform**(PlatformTO[] to) Inserta uno o varios Platform en la base de datos.
Result[]	**updatePlatform**(PlatformTO[] to) Actualiza los parámetros de uno o varios Platforms en la base de datos.

Constructor Detail

OraclePlatformDAO

```
public OraclePlatformDAO()
```

Constructor sin parámetros.

13.10.9 Clase OracleServiceDAO

```
java.lang.Object
  └ nagios.DAO.oracle.OracleServiceDAO
```

All Implemented Interfaces:
ServiceDAO

```
public class OracleServiceDAOextends java.lang.Objectimplements ServiceDAO
```

Clase encargada de la gestión de los objetos Services en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos, además de gestionar por separado sus atributos.

Author:

Agustín Bravo Ortiz

Constructor Summary

OracleServiceDAO() Constructor sin parámetros.	

Method Summary

Result[]	**deleteService**(ServiceTO[] to) Borra uno o varios Services de la base de datos.
Result	**deleteTrait**(java.lang.String typename, java.lang.String id_srv, Trait Attr) Borra un atributo de un Service de la tabla de atributos.
ServiceTO[]	**findService**(ServiceTO to) Busca Services en la base de datos usando un filtro de búsqueda, o muestra todos los Services existentes.
ServiceTO[]	**findTemplate**(ServiceTO to) Busca plantillas de Service en la base de datos, las cuales vendrán caracterizadas porque su atributo service_description comienza por "plnt_".
Result[]	**insertService**(ServiceTO[] to) Inserta uno o varios Services en la base de datos.
Result	**insertTrait**(java.lang.String typename, java.lang.String id_srv, Trait Attr) Inserta un atributo de un Service en la tabla de atributos.
ServiceTO	**showService**(ServiceTO to) Muestra todos los parámetros de un Service.
Result[]	**updateService**(ServiceTO[] to) Actualiza los parámetros de uno o varios Services.
Result	**updateTrait**(java.lang.String typename, java.lang.String id_srv, Trait Attr) Actualiza un atributo de un Service en la tabla de atributos.

Constructor Detail

OracleServiceDAO

```
public OracleServiceDAO()
```

Constructor sin parámetros.

13.10.10 Clase OracleServicegroupDAO

```
java.lang.Object
  └ nagios.DAO.oracle.OracleServicegroupDAO
```

All Implemented Interfaces:
ServicegroupDAO

```
public class OracleServicegroupDAOextends java.lang.Objectimplements
ServicegroupDAO
```

Clase encargada de la gestión de los objetos Servicegroups en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos, además de gestionar por separado sus atributos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleServicegroupDAO() Constructor sin parámetros.	

Method Summary

Result	**addService**(ServicegroupTO to) Añade un servicio al grupo de miembros del Servicegroup.
Result	**deleteService**(ServicegroupTO to) Elimina un servicio del grupo de miembros del Servicegroup.
Result[]	**deleteServicegroup**(ServicegroupTO[] to) Borra uno o varios Servicegroups de la base de datos.
ServicegroupTO[]	**findServicegroup**(ServicegroupTO to) Busca Servicegroups en la base de datos usando un filtro de búsqueda, o muestra todos los Servicegroups registrados.
Result[]	**insertServicegroup**(ServicegroupTO[] to) Inserta uno o varios Servicegroups en la base de datos.

ServicegroupTO	**showServicegroup**(ServicegroupTO to) Muestra un Servicegroup con todos sus parámetros.
Result[]	**updateServicegroup**(ServicegroupTO[] to) Actualiza los parámetros y miembros de uno o varios Servicegroups en la base de datos.

Constructor Detail

OracleServicegroupDAO

```
public OracleServicegroupDAO()
```

Constructor sin parámetros.

13.10.11 Clase OracleTimeperiodDAO

```
java.lang.Object
  └ nagios.DAO.oracle.OracleTimeperiodDAO
```

All Implemented Interfaces:
TimeperiodDAO

```
public class OracleTimeperiodDAOextends java.lang.Objectimplements TimeperiodDAO
```

Clase encargada de la gestión de los objetos Timeperiods en la base de datos. Se crearán, borrarán, actualizarán y se harán consultas de estos objetos.

Author:
Agustín Bravo Ortiz

Constructor Summary

OracleTimeperiodDAO() Constructor sin parámetros.	

Method Summary

Result[]	**deleteTimeperiod**(TimeperiodTO[] to) Borra uno o varios Timeperiods de la base de datos.
TimeperiodTO[]	**findTimeperiod**(TimeperiodTO to) Busca un Timeperiod a partir de un filtro de búsqueda, o muestra todos los existentes.
Result[]	**insertTimeperiod**(TimeperiodTO[] to) Inserta uno o varios Timeperiods en la base de datos.
Result[]	**updateTimeperiod**(TimeperiodTO[] to) Actualiza los atributos de uno o varios Timeperiods en la base de datos.

Constructor Detail

OracleTimeperiodDAO

```
public OracleTimeperiodDAO()
```

Constructor sin parámetros.

13.11 Paquete nagios.utilities

Class Summary

AutenticationFilter	Filtro que corre antes de cada petición a la aplicación y quese usa para comprobar que el usuario está en sesión.
Constants	Clase con las constantes necesarias para el funcionamiento del programa.
Controller	Clase que se encarga de la gestión de peticiones de acción de la aplicación.
ReqUtil	Clase que implementa distintos métodos que tratan con un objeto HttpServletRequest Se usa también como wrapper para objetos de tipo HttpServletRequest y MultipartParser, de forma que los métodos de acceso a sus atributos sean los mismos.
Result	Objeto que registra el resultado de una operación.
Trait	Objeto que almacena el valor del atributo.
Utilities	Cajón de sastre para distinto métodos útiles que son usados por

	diferentes clases

13.11.1 Clase AutenticationFilter

```
java.lang.Object
  └nagios.utilities.AutenticationFilter
```

All Implemented Interfaces:
javax.servlet.Filter

```
public class AutenticationFilterextends java.lang.Objectimplements
javax.servlet.Filter
```

Filtro que corre antes de cada petición a la aplicación y que se usa para comprobar que el usuario está en sesión.

Constructor Summary

AutenticationFilter()	

Method Summary

void	**destroy**()
void	**doFilter**(javax.servlet.ServletRequest request, javax.servlet.ServletResponse response, javax.servlet.FilterChain chain)
void	**init**(javax.servlet.FilterConfig filterConfig)

Constructor Detail

AutenticationFilter

```
public AutenticationFilter()
```

13.11.2 Clase Constants

Field Summary	
static java.lang.String	**ACTION** Acción que se va a ejecutar.
static java.lang.String	**ACTIVE** Estado activo.
static java.lang.String	**ADMUSER** Nombre con el que se guarda el objeto que representa al usuario administrador en la sesión.
static java.lang.String	**APLICATIONNAME** Nombre de la aplicación, necesario para insertarlo en la dirección con la que se ataca al servidor de autenticación.
static java.lang.String	**CFGCABECERA** Cabecera de comienzo de archivo de configuración generado por la aplicación.
static java.lang.String	**CFGCOMMANDS** Archivo de configuración para los checkcommands.
static java.lang.String	**CFGCONTACTGROUPS** Archivo de configuración para los contactgroups.
static java.lang.String	**CFGCONTACTS** Archivo de configuración para los contacts.
static java.lang.String	**CFGHOSTDEPEDECIES** Archivo de configuración para los Hostdependencies.
static java.lang.String	**CFGHOSTESCALATIONS** Archivo de configuración para los Hostescalations.
static java.lang.String	**CFGHOSTGROUPS** Archivo de configuración para los Hostgroups.
static java.lang.String	**CFGHOSTS** Archivo de configuración para los Hosts.
static java.lang.String	**CFGMISCCOMMANDS** Archivo de configuración para los checkcommands destinados a notificación.
static java.lang.String	**CFGNAGIOS**

	Ruta del archivo principal de configuración de Nagios.
static java.lang.String	**CFGPATH** Path en el que se encuentran todos los archivos de configuración.
static java.lang.String	**CFGPATHSAVE** Path en el que se guardan los archivos .zip con las configuraciones antiguas.
static java.lang.String	**CFGSERVICEDEPENDENCIES** Archivo de configuración para los Servicedependencies.
static java.lang.String	**CFGSERVICEESCALATIONS** Archivo de configuración para los Serviceescalations.
static java.lang.String	**CFGSERVICEGROUPS** Archivo de configuración para los Servicegroups.
static java.lang.String	**CFGSERVICES** Archivo de configuración para los Services.
static java.lang.String	**CFGTIMEPERIODS** Archivo de configuración para los Timeperiods.
static java.lang.String	**COMMANDS** Checkcommands existentes.
static java.lang.String	**CONTACTGROUPS** Contactgroups existentes.
static java.lang.String	**CONTACTS** Contacts existentes.
static java.lang.String	**DATA** Nombre con el que se insertan los datos esenciales de determinada consulta.
static java.lang.String	**ERROR** Error que se ha producido durante la ejecución.
static java.lang.String	**ERROR_URLAUTENTICATION** Página de la Junta a la que se redirecciona en caso de producirse un error en la autenticación con la firma.
static java.lang.String	**FORMATOFECHA_ORA** Formato de fecha simple para oracle.
static java.lang.String	**FORMATOFECHAHORA_ORA** Formato de fecha con hora para oracle.
static java.lang.String	**HOSTGROUPS** Hostgroups existentes.

static java.lang.String	**HOSTS** Hosts existentes.
static java.lang.String	**INACTIVE** Estado inactivo.
static java.lang.String	**LOG4JPATH** Path en el que se guarda el log.
static java.lang.String	**NAGIOS** Ruta del programa Nagios.
static int	**OPERACION_OK** Operación correcta.
static int	**OPERATION_FAILURE** Operación fallida.
static java.lang.String	**OPERATION_RESULT** Resultado de la operación.
static java.lang.String	**ORACLEJNDI** Conector jdbc para el acceso a la base de datos.
static java.lang.String	**PLATFORMS** Plataformas existentes.
static java.lang.String	**SEPARADOR** Barra usada en la construcción del Path de los archivos.
static java.lang.String	**SERVICES** Services existentes.
static int	**SQL_DUPLICATE_ERROR** Fila duplicada.
static int	**SQL_ERR_PKDUPLICADA_VALORESDISTINTOS** Fila duplicada con valores distintos.
static int	**SQL_GENERAL_ERROR** Error genérico.
static int	**SQL_LONG_ERROR** El campo excede el tamaño asignado a la columna en la tabla.
static int	**SQL_OK** Operación correcta.
static int	**SQL_PKNOFIND_ERROR** No se encuentra la clave foránea.
static java.lang.String	**TEMPLATE** Plantilla que se va a utilizar.

static java.lang.String	**TIMEPERIODS** Timeperiods existentes.
static java.lang.String	**TMPLAVALIABLES** Plantillas disponibles de un determinado objeto.
static java.lang.String	**URLAUTENTICATION** URL que se usa para la autenticación.

13.11.3 Clase Controller

```
java.lang.Object
  └javax.servlet.GenericServlet
      └javax.servlet.http.HttpServlet
          └nagios.utilities.Controller
```

All Implemented Interfaces:
java.io.Serializable, javax.servlet.Servlet, javax.servlet.ServletConfig

---

```
public class Controllerextends javax.servlet.http.HttpServlet
```

Clase que se encarga de la gestión de peticiones de acción de la aplicación. Sobrescribe los método doGet y doPost y en estos se encarga de lanzar la ejecución de la acción, recoger la vista a la que hay que redireccionar y todos los demás objetos (si los ha generado la ejecución de la acción) para insertarlos en el request, y por último realizar la redirección a la vista o a las páginas de resultados o de error según corresponda.

See Also:
Serialized Form

Constructor Summary

Controller()	

Method Summary

void	**destroy**()
void	**doGet**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)

void	**doPost**(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse response)
void	**init**(javax.servlet.ServletConfig config)

Methods inherited from class javax.servlet.http.HttpServlet
service

Methods inherited from class javax.servlet.GenericServlet
getInitParameter, getInitParameterNames, getServletConfig, getServletContext, getServletInfo, getServletName, init, log, log

Constructor Detail

Controller

```
public Controller()
```

Method Detail

init

```
public void init(javax.servlet.ServletConfig config)
```

Specified by:
`init` in interface `javax.servlet.Servlet`
Overrides:
`init` in class `javax.servlet.GenericServlet`

---

doGet

```
public void doGet(javax.servlet.http.HttpServletRequest request,
                  javax.servlet.http.HttpServletResponse response)
           throws javax.servlet.ServletException,
                  java.io.IOException
```

Overrides:
`doGet` in class `javax.servlet.http.HttpServlet`
Throws:

javax.servlet.ServletException
java.io.IOException

---

doPost

```
public void doPost(javax.servlet.http.HttpServletRequest request,
                   javax.servlet.http.HttpServletResponse response)
```

Overrides:
doPost in class javax.servlet.http.HttpServlet

---

destroy

```
public void destroy()
```

Specified by:
destroy in interface javax.servlet.Servlet
Overrides:
destroy in class javax.servlet.GenericServlet

13.11.4 Clase ReqUtil

```
java.lang.Object
  └nagios.utilities.ReqUtil
```

```
public class ReqUtilextends java.lang.Object
```

Clase que implementa distintos métodos que tratan con un objeto HttpServletRequest Se usa también como wrapper para objetos de tipo HttpServletRequest y MultipartParser, de forma que los métodos de acceso a sus atributos sean los mismos. Esto es así porque el constructor MultipartParser solo puede llamarse una vez, por lo que hemos de recoger todos sus parámetros en ese momento y almacenarlos para su posterior uso.

Constructor Summary	
ReqUtil(javax.servlet.http.HttpServletRequest req)	

Constructor de la clase con un objeto HttpServletRequest

Method Summary

Action	**getAccion**() Instancia la acción correspondiente a la llamada
com.oreilly.servlet.multipart.FilePart	**getFichero**(java.lang.String nombre) Obtiene el fichero de la subida
java.lang.String	**getParameter**(java.lang.String param) Wrapper de request.getParameter Obtiene valor asociado al parametro.
javax.servlet.http.HttpServletRequest	**getRequest**()
void	**showValues**() Muestra los valores almacenados o recoge los valores si no hay y los muestra en el log de la aplicación.

Constructor Detail

ReqUtil

```
public ReqUtil(javax.servlet.http.HttpServletRequest req)
        throws javax.servlet.ServletException
```

Constructor de la clase con un objeto HttpServletRequest

Parameters:
req -
Throws:
javax.servlet.ServletException

Method Detail

getRequest

```
public javax.servlet.http.HttpServletRequest getRequest()
```

Returns:
the request

getParameter

```
public java.lang.String getParameter(java.lang.String param)
```

Wrapper de request.getParameter Obtiene valor asociado al parametro.

Parameters:
`param` - Parámetro del que queremos obtener su valor
Returns:
Valor asociado del parámetro

---

getFichero

```
public com.oreilly.servlet.multipart.FilePart
getFichero(java.lang.String nombre)
```

Obtiene el fichero de la subida

Parameters:
`nombre` -
Returns:
Nombre del fichero.

---

getAccion

```
public Action getAccion()
```

Instancia la acción correspondiente a la llamada

Returns:
Objeto acción que devuelve el método createAccion

---

showValues

```
public void showValues()
```

Muestra los valores almacenados o recoge los valores si no hay y los muestra en el log de la aplicación.

13.11.5 Clase Result

```
java.lang.Object
  └nagios.utilities.Result
```

```
public class Resultextends java.lang.Object
```

Objeto que registra el resultado de una operación. Será utilizado para el registro de errores y la información detallada de resultados.

Constructor Summary

Result() Constructor sin parámetros.	
Result(int codigo) Constructor en el que se fija el código.	

Method Summary

int	**getCode**()
java.lang.String	**getDescription**()
java.lang.String	**getSituation**()
void	**setCode**(int codigo)
void	**setDescription**(java.lang.String descripcion)
void	**setSituation**(java.lang.String situation)

Constructor Detail

Result

```
public Result()
```

Constructor sin parámetros.

Result

```
public Result(int codigo)
```

Constructor en el que se fija el código.

Parameters:
`codigo` - Valor que se asigna al código.

Method Detail

getDescription

```
public java.lang.String getDescription()
```

Returns:
Descripción del resultado.

---

getCode

```
public int getCode()
```

Returns:
the codigo

---

setCode

```
public void setCode(int codigo)
```

Parameters:
`codigo` - the codigo to set

---

getSituation

```
public java.lang.String getSituation()
```

Returns:
the situation

---

setSituation

```
public void setSituation(java.lang.String situation)
```

Parameters:
`situation` - the situation to set

---

setDescription

```
public void setDescription(java.lang.String descripcion)
```

Parameters:
`descripcion` - the descripcion to set

13.11.6 Clase Trait

```
java.lang.Object
  └nagios.utilities.Trait
```

```
public class Traitextends java.lang.Object
```

Author:
agustin Esta clase representa un atributo de un objeto de configuración de nagios. Además del valor tendremos otros parámetros asociados. Usamos sobreescritura de métodos para que el valor sea un String, un boolean o un int.

Constructor Summary	
Trait() Constructor sin parámetros.	
Trait(boolean value) Constructor con el que se establece valor como un booleano y override a true.	

Trait(boolean value, boolean override) Constructor con todos los parámetros.	
Trait(java.lang.Object[] value) Constructor en el que se establece value como un String de objetos, usualmente de tipo String, y override siempre a true.	
Trait(java.lang.Object[] value, boolean override) Constructor con todos los parámetros.	
Trait(java.lang.String value) Constructor con el que se establece valor como un String y override a true.	
Trait(java.lang.String value, boolean override) Constructor con todos los parámetros.	

Method Summary	
java.lang.Object	**getValue**()
boolean	**isOverride**()
void	**setOverride**(boolean override)
void	**setValue**(java.lang.Object value)

13.11.7 Clase Utilities

```
java.lang.Object
  └ nagios.utilities.Utilities
```

---

public class **Utilities**extends java.lang.Object

Cajón de sastre para distinto métodos útiles que son usados por diferentes clases

---

Constructor Summary	
Utilities()	

Method Summary	
static boolean	**checkFormatPeriod**(java.lang.String periodo) Comprueba que el formato del periodo definido es el correcto para la configuración de Nagios.
static java.lang.String	**ejecuta**(java.lang.String comando) Método para ejecutar un comando en el servidor.
static boolean	**eliminaFichero**(java.lang.String path) Elimina un fichero
static int	**esEntero**(java.lang.String cadena) Comprueba que la cadena pasada tiene formato de entero POSITIVO
static java.lang.String[]	**getFichero**(java.lang.String nombre) Devuelve el nombre de un fichero y su extensión, es decir lo que esta detrás del '.'
static java.lang.String	**getTextoCodSQL**(int codigo) Devuelve el texto asociado a un código SQL (normalmente errores)
static java.lang.String[]	**listDir**(java.lang.String path) Lista los nombres de los archivos de un directorio
static boolean	**mueveFichero**(java.lang.String pathOrigen, java.lang.String pathDestino) Mueve un fichero de un PATH a otro Si el fichero ya existe, lo machaca
static boolean	**NIFcorrecto**(java.lang.String nif) Comprueba que el NIF es correcto
static boolean	**StrToBoolean**(java.lang.String param) Convierte de String a Boolean
static java.lang.String[]	**unzip**(java.lang.String pathZip, java.lang.String pathDestino) Descomprime un .zip
static Result	**zip**(java.lang.String pathZip, java.lang.String[] files) Comprime un conjunto de archivos en un archivo .zip

Constructor Detail

Utilities

```
public Utilities()
```

Method Detail

getFichero

```
public static java.lang.String[] getFichero(java.lang.String nombre)
```

Devuelve el nombre de un fichero y su extensión, es decir lo que esta detrás del '.'

Parameters:
`nombre` - Nombre del fichero
Returns:
array[0] = nombre del fichero o null si no tiene; array[1] = extensión del fichero o null si no tiene

---

getTextoCodSQL

```
public static java.lang.String getTextoCodSQL(int codigo)
```

Devuelve el texto asociado a un código SQL (normalmente errores)

Parameters:
`codigo` - Valor del que se quiere obtener el texto asociado.
Returns:
Texto asociado al código SQL

NIFcorrecto

```
public static boolean NIFcorrecto(java.lang.String nif)
```

Comprueba que el NIF es correcto

Parameters:
`nif` - NIF a comprobar
Returns:
Resultado de la comprobación

---

esEntero

```
public static int esEntero(java.lang.String cadena)
```

Comprueba que la cadena pasada tiene formato de entero POSITIVO

Parameters:
`cadena` - Cadena a comprobar
Returns:
Longitud de la cadena □ -1 si la cadena no tiene formato numérico

---

mueveFichero

```
public static boolean mueveFichero(java.lang.String pathOrigen,
                                   java.lang.String pathDestino)
```

Mueve un fichero de un PATH a otro Si el fichero ya existe, lo machaca

Parameters:
`pathOrigen` - Fichero que se quiere mover
`pathDestino` - Destino al que se quiere mover el fichero.
Returns:
true si ha ido todo bien. False en caso contrario

---

eliminaFichero

```
public static boolean eliminaFichero(java.lang.String path)
```

Elimina un fichero

Parameters:
`path` - Fichero que se quiere eliminar
Returns:
true si ha ido todo bien. False en caso contrario

---

listDir

```
public static java.lang.String[] listDir(java.lang.String path)
```

Lista los nombres de los archivos de un directorio

Parameters:

`path` - Ruta del directorio a escanear

Returns:

Array con la lista de nombres de archivos

---

zip

```
public static Result zip(java.lang.String pathZip,
                         java.lang.String[] files)
```

Comprime un conjunto de archivos en un archivo .zip

Parameters:

`pathZip` - Ruta del archivo .zip que se genera

`files` - Array con los path completos de los archivos a comprimir

Returns:

Objeto Resultado en el que se informa sobre el resultado de la operación.

Throws:

`java.io.FileNotFoundException`

`java.io.IOException`

---

unzip

```
public static java.lang.String[] unzip(java.lang.String pathZip,
                                       java.lang.String pathDestino)
```

Descomprime un .zip

Parameters:

`pathZip` - Ruta del fichero zip que queremos descomprimir

`pathDestino` - Ruta en donde queremos que se descomprima el fichero

Returns:

array con los nombres de los ficheros descomprimidos incluidos en el .zip

---

checkFormatPeriod

```
public static boolean checkFormatPeriod(java.lang.String periodo)
```

Comprueba que el formato del periodo definido es el correcto para la configuración de Nagios. Este formato es del tipo: 12:00-13:00,14:00-15:00,...

Parameters:
`periodo` - String con el periodo que se quiere comprobar.
Returns:
true si el formato es correcto o false si es incorrecto.

StrToBoolean

```
public static boolean StrToBoolean(java.lang.String param)
```

Convierte de String a Boolean

Parameters:
`param` - String que se quiere convertir.
Returns:
true si la cadena de entrada tiene como valor: "true", "ovrd" o "1". En caso contrario se devuelve false.

---

ejecuta

```
public static java.lang.String ejecuta(java.lang.String comando)
                                throws java.io.IOException
```

Método para ejecutar un comando en el servidor. Devuelve la respuesta del sistema.

Parameters:
`comando` - comando que se ejecuta
Returns:
Respuesta
Throws:
`java.io.IOException`