DESARROLLO DE UNA HERRAMIENTA PARA LA ADMINISTRACIÓN DE LA SEGURIDAD EN APLICACIONES EMPRESARIALES, BASADAS EN EL *FRAMEWORK ACEGI*

Ing. Fidel Alejandro Ortega Orihuela[1], Dr C Jorge D. Ortega Suárez[2]

1. Instituto Politécnico "Carlos Marx"

2. Centro de Estudios del Medio Ambiente de Matanzas, Facultad de Ingenierías Química y Mecánica, Universidad de Matanzas "Camilo Cienfuegos"; Carr. a Varadero km 3 Matanzas, Cuba.

Resumen.

La administración de la seguridad en aplicaciones que se desarrollan sobre la plataforma *Java Enterprise Edition* (JEE), frecuentemente se hace mediante la utilización del *framework Acegi* o *Spring Security* como se nombra actualmente a partir de su última versión oficial.

A pesar de la gran cantidad de ventajas y utilidades que proporciona *Acegi*, este presenta una complicación importante, reflejada en el hecho de que toda la configuración de sus reglas y políticas de seguridad para una aplicación, se hace de forma manual en ficheros XML.

Este trabajo propone como solución a esta problemática el desarrollo de una aplicación que permita administrar de forma fácil y eficiente estas configuraciones. Una herramienta que automatice este proceso ahorrará mucho más tiempo en cualquier proceso de desarrollo de software así como disminuirá los errores involuntarios en la configuración de las reglas. La metodología de desarrollo utilizada fue *eXtreme programming* (XP), lo que permitió construir un producto sencillo, intuitivo y fácil de usar. Se introduce actualmente en la gestión estratégica de la Corporación CubaRon S.A, además de ser utilizado en algunos proyectos productivos de la Universidad de las Ciencias Informáticas.

***Palabras claves**: Java Enterprise Edition, Framework Spring, Framework Acegi, Configuración de la seguridad.*

Introducción.

Gestionar las políticas de seguridad de una aplicación es un aspecto que afecta a prácticamente la totalidad de las aplicaciones empresariales, entendiendo por seguridad la necesidad de saber que el usuario es quien dice ser (autenticación), y permitirle acceso sólo a aquellos recursos necesarios (autorización)[1]. La implementación de tales políticas es una tarea complicada y resulta muchas veces en código ligado con las funciones de negocio. Si no se adoptan desde una perspectiva correcta, pueden llegar a ser una carga que afectará y lastrará el desarrollo del sistema[1].

Las políticas de seguridad concernientes a una aplicación, con frecuencia cambian durante su implementación, lo que obliga a los programadores a actualizarlas constantemente. El presente trabajo ofrece una herramienta para configurar con eficacia tales políticas de seguridad en aplicaciones empresariales, basada en el *framework Acegi*

Desarrollo.

Las aplicaciones empresariales se desarrollan en diferentes lenguajes y plataformas. El lenguaje de programación *Java* es actualmente uno de los más utilizados en la creación de *software* de empresa en el mundo. La plataforma Java ha atraído alrededor de 4 millones de desarrolladores de *software*, se utiliza en los principales sectores de la industria de todo el planeta y está presente en un gran número de dispositivos, ordenadores y redes de cualquier tecnología de programación. De hecho, su versatilidad, eficiencia, portabilidad y la seguridad que aporta, la han convertido en la tecnología ideal para su aplicación a redes, de manera que hoy en día, más de 2.500 millones de dispositivos la emplean[2].

Con su evolución se han desarrollado tres plataformas, cada una de ellas orientada a cubrir un entorno diferente:

- *Java Standard Edition* (JSE), colección de APIs del lenguaje de programación Java útiles para muchos programas de la plataforma.

- *Java Micro Edition* (JME), colección de APIs de Java para el desarrollo de software para dispositivos de recursos limitados, como PDA, teléfonos móviles y otros aparatos de consumo.

- *Java Enterprise Edition* (JEE), plataforma para crear aplicaciones cliente-servidor.

La plataforma JEE es la edición empresarial de la plataforma *Java* y está especialmente pensada para la creación de aplicaciones *web*[3]. Aprovecha las fortalezas de la edición estándar de *Java* (J2SE), complementándolas con especificaciones, funcionalidades y lineamientos orientados al desarrollo de aplicaciones empresariales. Su nombre original era *Java 2 Enterprise Edition* (J2EE), sin embargo, a partir de la edición 5 se cambió a JEE. De manera general el lenguaje *Java* se apoya en la utilización de APIs, *frameworks* y librerías, que proporcionan recursos reutilizables y funcionalidades que permiten ahorrar tiempo de trabajo a los desarrolladores.

La plataforma JEE cuenta con varios de estos elementos, entre ellos, un grupo significativo de *frameworks* especializados. Dentro de estos, se encuentra el *framework* de código abierto *Spring* para el desarrollo de aplicaciones en la plataforma *Java* EE. La

primera versión del mismo fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro —*Expert One-on-One Java EE Design and Development*, por la editorial *Wrox Press* en octubre del año 2002[4].

El centro de *Spring* se basa en el principio de *Inversion of Control* (IoC) o inyección de dependencias. Esta técnica hace externa la creación y el manejo de las dependencias de los componentes, logrando mayor limpieza y claridad en el código pues provee, en tiempo de ejecución, de todas las instancias de clases de la aplicación y las dependencias que ellas necesitan[5] [6]. Con este mecanismo se obtienen los siguientes resultados:

- Reduce potencialmente el código de enlace entre los diferentes componentes de la aplicación.

- Externaliza las dependencias, lo cual permite la reconfiguración de las mismas sin necesidad de compilar todo el código.

- Maneja las dependencias en un solo lugar, facilitando la configuración de las mismas y disminuyendo el margen de errores.

- Fomenta un buen diseño de la aplicación, debido a que la inyección de instancias está basada en interfaces, y las clases que las implementan son creadas por el contenedor de IoC.

- Además de ser IoC una de las características más relevantes de Spring, se encuentra el soporte a la AOP, la cual es una de las tecnologías del momento en el mundo de *Java*. AOP permite efectuar procesamientos comunes en muchas partes de la aplicación (*crosscutting*) implementándolos en un solo lugar. En *Spring*, AOP es usado para muchos propósitos como el manejo de transacciones, manejo de trazas, seguridad, y permite hacerlo en muchos casos de forma declarativa [5] [6].

Después de ver algunas características y ventajas que brinda el *framework Spring*, puede apreciarse que este facilita la construcción de aplicaciones Java. A diferencia del *framework Struts*, *Spring* se puede utilizar en cualquier tipo de aplicación, es ligero por el mínimo impacto que tiene en las aplicaciones y trae integrado el *framework Acegi*.

Acegi Security System[7] es un *framework* creado por Ben Alex en el año 2003, liberado bajo la licencia de Apache en el 2004 e íntimamente ligado al proyecto *Spring*[8]. En el año 2008 - y después de dos años de desarrollo -, la plataforma de aplicaciones para *Java Spring Source* ha terminado la nueva versión de dicho *framework*, que implica la integración completa con *Spring* y el cambio de nombre de *Acegi Security System* a *Spring Security*.

Este *framework* facilita la tarea de adoptar medidas de seguridad en aplicaciones *Java*. Además, combinado con AOP y la inyección de instancias de *Spring* (IoC), brinda un mecanismo de seguridad potente que permite definirlo de manera declarativa, transparente para el desarrollador y sin necesidad de escribir código nuevo, utilizando para ello, el soporte prestado por el *framework Spring*, pero siendo posible utilizarlo en aplicaciones no desarrolladas con *Spring*[1].

Acegi permite mantener la lógica de negocio libre de código de seguridad, es *open-source*, sin coste de licencias, con el respaldo de un enorme y creciente grupo de usuarios que lo utilizan, además de poseer un manual de referencia con más de 90 páginas que no tiene nada que envidiar a la documentación de un producto comercial[1]. La arquitectura de *Acegi* está fuertemente basada en interfaces y en patrones de diseño, proporcionando las implementaciones más comúnmente utilizadas, y numerosos puntos de extensión donde nuevas funcionalidades pueden ser añadidas sin tocar el código existente. Al utilizar esta arquitectura puede resultar un poco difícil seguir el flujo de ejecución al principio, pero una vez comprendida la idea global, se acepta como el precio necesario de disfrutar un *framework* con una gran potencia[1].

Si bien existe el estándar *Java Authorization and Authentication Service* (JAAS) que pretende cubrir tanto autenticación como autorización, su adopción dista mucho de ser sencilla y portable, debido a que el soporte proporcionado por los contenedores de aplicaciones está lejos de ser adecuado, existen incompatibilidades entre distintas implementaciones y cada contenedor requiere una configuración distinta, normalmente con adición de librerías[1]. Esta configuración se realiza en los contenedores de aplicaciones y/o en la máquina virtual. Cada vendedor de un servidor de aplicación es libre de implementar la seguridad de forma diferente y no se requiere necesariamente la utilización de JAAS. Además, este estándar no considera la seguridad de la manera en que realmente es, como uno de los aspectos más trabajosos y difíciles dentro del desarrollo de un producto de *software*[9].

La funcionalidad y soporte proporcionados por *Acegi* son mucho mayores que los de JAAS y además permite la integración con este, utilizándolo en la fase de autenticación[1]. *Acegi* proporciona cuatro opciones principales de seguridad:

- Listas de control de acceso (ACL) *web* basadas en esquemas URL[10].
- Protección de métodos y clases *Java* utilizando AOP[10].
- *Single Sign-On* (SSO ACL)[10].
- Seguridad proporcionada por el contenedor *Web*[10].

Acegi está formado, a su vez, por cinco componentes:

- *Security Interceptor*: Previene el acceso a los recursos seguros de la aplicación. Delega el manejo de la seguridad a los restantes componentes.
- *Authentication Manager*: Asume el proceso de autenticación. Es una interfaz conectada, basada en componentes, lo que hace posible la utilización de Acegi, virtualmente con cualquier mecanismo de seguridad imaginable.
- *Access Decision Manager*: Es el encargado de decidir si un usuario puede acceder o no a un recurso determinado.
- *Run-As Manager*: Permite remplazar los permisos para recursos más profundos en la aplicación. Constituye un componente opcional en el manejo de la seguridad con *Acegi*.

- *After-invocation Manager*. Se encarga de reforzar la seguridad de los recursos de la aplicación, una vez que se ha accedido a ellos.

Como se ha visto la gestión de la seguridad lleva consigo dos procesos:

1. Autenticación, consistente en determinar la identidad del usuario, generalmente mediante un nombre de usuario y una contraseña.

2. Control del acceso, consistente en determinar si un usuario previamente autenticado tiene acceso al recurso solicitado.

Acegi realiza su trabajo de forma robusta, elegante y de manera poco intrusiva, además de permitir la integración con el resto de los mecanismos. La configuración de las reglas y políticas de seguridad que *Acegi* proporciona es almacenada en ficheros XML, y la edición de estos se hace de forma manual.

Es un hecho que en aplicaciones incluso de pequeño a mediano tamaño, los ficheros de configuración tienden a hacerse muy extensos y poco legibles. La introducción de errores involuntarios por parte de los responsables de su configuración, o la simple pérdida de tiempo en el proceso, obstaculizan el trabajo de los desarrolladores durante la fase de implementación, respecto al rendimiento y el tiempo de desarrollo. Esa necesidad creada demanda una solución eficaz correspondiente, a saber, la de crear una herramienta que permita resolver de forma automatizada el proceso de administración de las políticas de seguridad proporcionadas por *Acegi* para aplicaciones que se desarrollen con tecnología *Java* EE.

Descripción de la herramienta propuesta. Metodología utilizada.

La metodología empleada para el desarrollo de la herramienta propuesta como solución al problema de la investigación fue XP (e*Xtreme Programing*)[11]. XP se basa en *UseStories* (historias de usuario), las cuales son escritas por el cliente o su representante dentro del equipo y describen los escenarios claves del funcionamiento del *software*. Estas historias son el artefacto más importante generado por XP y cumplen un objetivo muy similar al de los casos de uso en metodologías pesadas como RUP. La posibilidad de hablar en el mismo idioma que el cliente, permite una comunicación y un entendimiento más rápido y fluido entre las partes involucradas.

A partir de estas historias se generan los *releases* (entregas) entre el equipo y el cliente. Los *releases* son los que permiten definir las iteraciones necesarias para cumplir con los objetivos, de manera que cada resultado de la iteración sea un programa aprobado por el cliente de quien depende la definición de las posteriores iteraciones.

***Integrated Development Enviroment* (IDE) y lenguaje de programación**.

El IDE de desarrollo seleccionado para desarrollar la herramienta fue el *wxDev*-C++, el cual utiliza como lo indica su nombre un compilador de C++. Una de las razones que motivaron su elección fue que tanto el IDE como el compilador son *open-source* y gratis, además de estar basado en *wxWidgets*, lo que le permite ofrecer portabilidad a un gran número de plataformas. Otra de las razones que lo avalan es que utiliza el lenguaje de programación C++, el cual es muy potente y uno de los más robustos y completos que se hayan conocido.

***Computer Aided Software Engineering* (CASE)**.

La herramienta CASE seleccionada para la creación del diagrama de clases del diseño, fue la *ArgoUML* por ser *open-source* y soportar el estándar UML 1.4.

Arquitectura de la aplicación

La herramienta desarrollada está basada en un modelo de aplicación en tres capas, a partir del cual se separaron conceptualmente los procesos a modelar en tres grupos, en dependencia de las características y responsabilidades de cada uno de ellos. Una arquitectura en tres capas se basa en la separación del sistema en tres capas lógicas, las cuales mantienen una estrecha relación de comunicación entre sí pero, a su vez, mantienen una total independencia al cumplir con sus responsabilidades, sólo apoyándose entre ellas para desarrollar tareas cuyas actividades requieran ser procesadas por más de una capa.

A partir de estos tres grupos de procesos, se definieron un conjunto de clases encargadas de manipular y controlar cada una de las tres capas lógicas del sistema; la capa de presentación, la de negocio y la capa de acceso a datos.

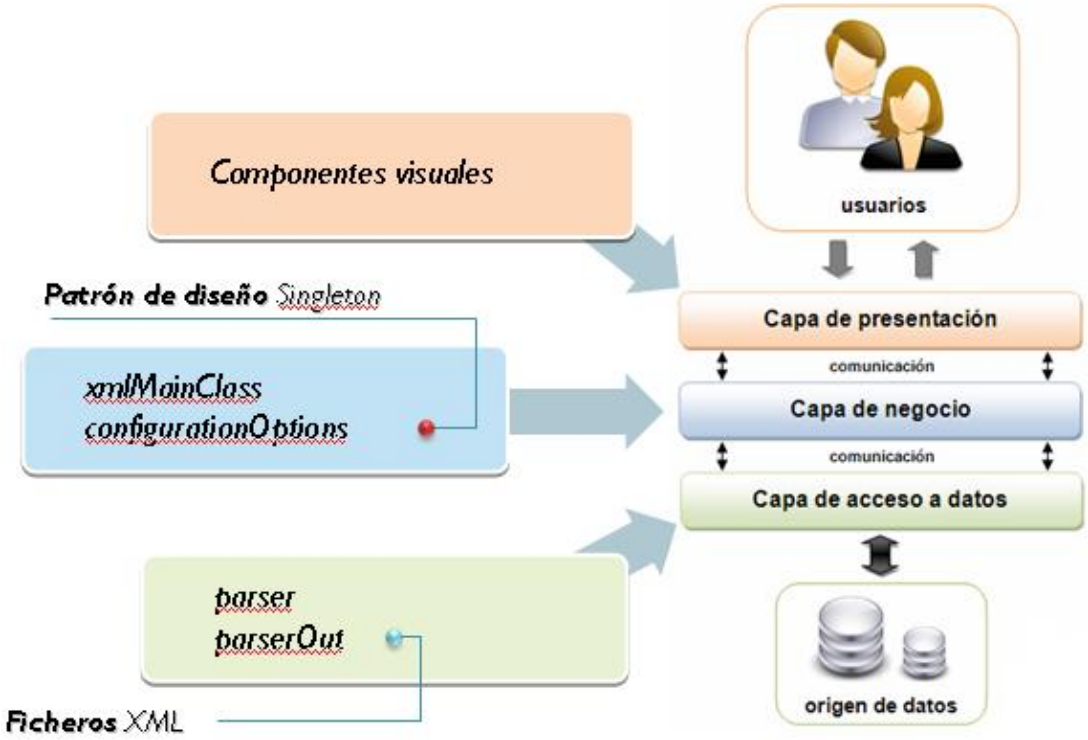

Figura 1: Separación lógica en capas.

Patrones de diseño empleados en la herramienta

Sólo es utilizado un patrón de diseño en esta solución, el patrón *Singleton* (instancia única), y es empleado específicamente en este *software* para proveer una única instancia de la clase que contiene las configuraciones generales accedidas en numerosos puntos de la aplicación. Esto garantiza que el objeto accedido es siempre el mismo, sin importar el punto o el momento de acceso, evitando además que este deba ser pasado

constantemente como parámetro a clases que lo necesiten. La implementación de este patrón se hace agregando en la clase un método estático encargado de crear una instancia de la misma sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada de otra forma que no sea el uso de este método, se regula el alcance del constructor (haciendo al mismo privado o protegido en la clase).

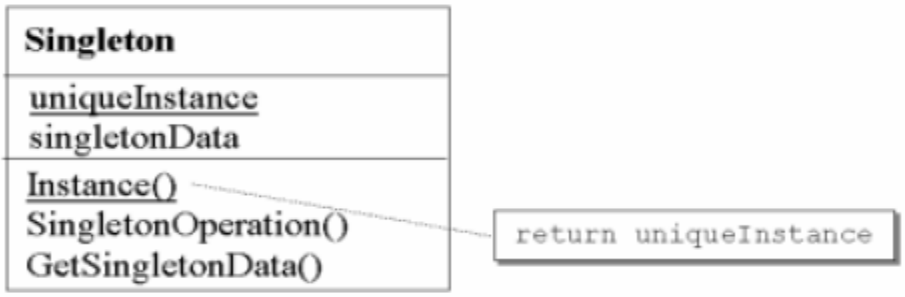

Figura 2: Estructura general del patrón Singleton.

Historia de Usuario	
Número: 1	**Nombre historia:** Configurar *Authentication Managers*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente "competente" en el equipo.	**Iteración asignada:** 1
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Fidel Alejandro Ortega Orihuela	
Descripción Configurar la autenticación, debe permitir la configuración del nodo <*bean*> principal, con todas las propiedades y requisitos necesarios, de forma fácil y sugerente. Aún cuando se cuente con la aplicación para la configuración del XML de forma visual, no debe obviarse la posibilidad de mantener lo más legible posible este último, pues por alguna u otra razón puede ser necesaria la edición manual del mismo. Lo anterior puede lograrse mediante la inclusión de comentarios, líneas en blanco entre los nodos, y una tabulación correcta.	
Observaciones:	

Figura 3: Historia de usuario #1 de la herramienta propuesta

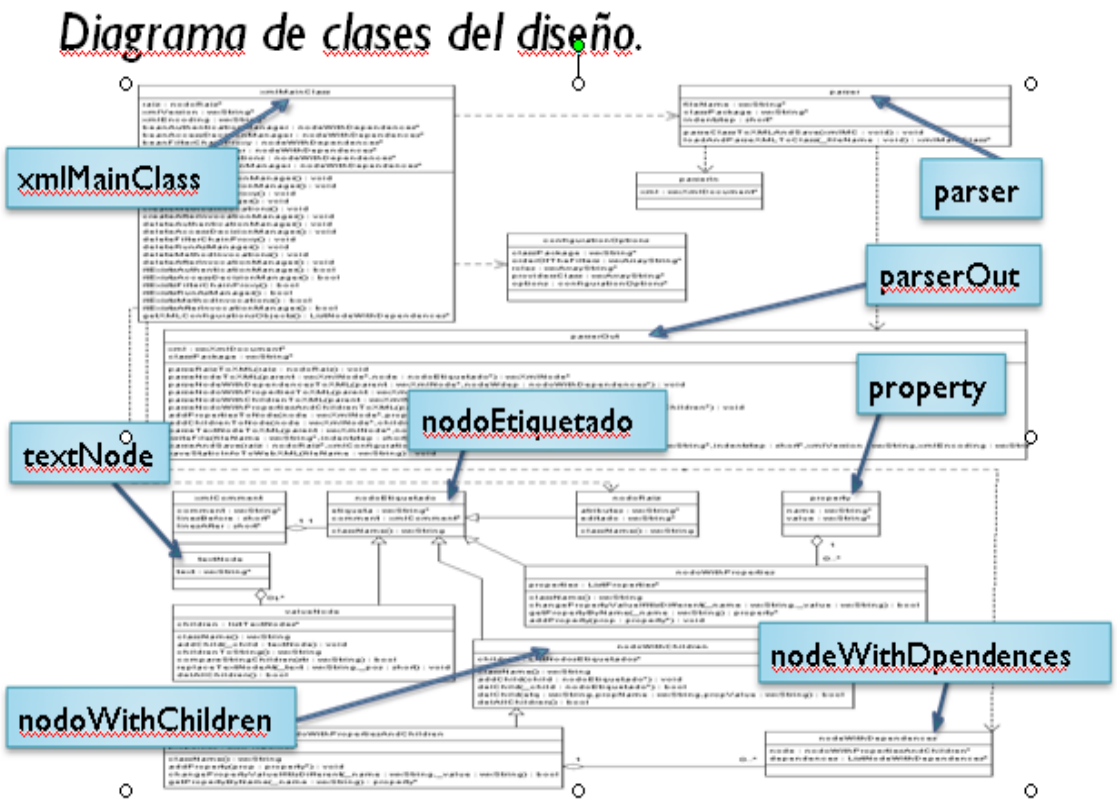

Figura 4: Diagrama de clases del diseño de la aplicación

Descripción de las clases del diseño.

Nombre	xmlMainClass
Tipo de clase	Controladora
Descripción	
Es el centro de control de la aplicación, contiene y manipula los atributos que hacen persistir los datos en memoria mientras esta está en ejecución, se encarga de iniciar todas las operaciones que deben ser ejecutadas en el momento que el usuario se lo solicita a través de los controles gráficos de las clases interfaz.	
Atributo	**Tipo**
raiz	nodoRaiz*
xmlVersion	wxString*
xmlEncoding	wxString*
beanAuthenticationManager	nodeWithDependences*
beanAccessDecisionManager	nodeWithDependences*
beanFilterChainProxy	nodeWithDependences*
beanRunAsManager	nodeWithDependences*
beanMethodInvocations	nodeWithDependences*
beanAfterInvocationManager	nodeWithDependences*
Para cada responsabilidad	
Nombre	void createAuthenticationManager()
Descripción	Es el responsable de crear el objeto (nodo bean) AuthenticationManager, que contiene toda la configuración referente a la autenticación, es el nodo principal de la autenticación y contiene una lista de todos los nodos que colaboran con él.
Nombre	void createAccessDecisionManager()

Figura 5: Descripción de la clase controladora *xmlMainClass*

Interfaz gráfica del sistema

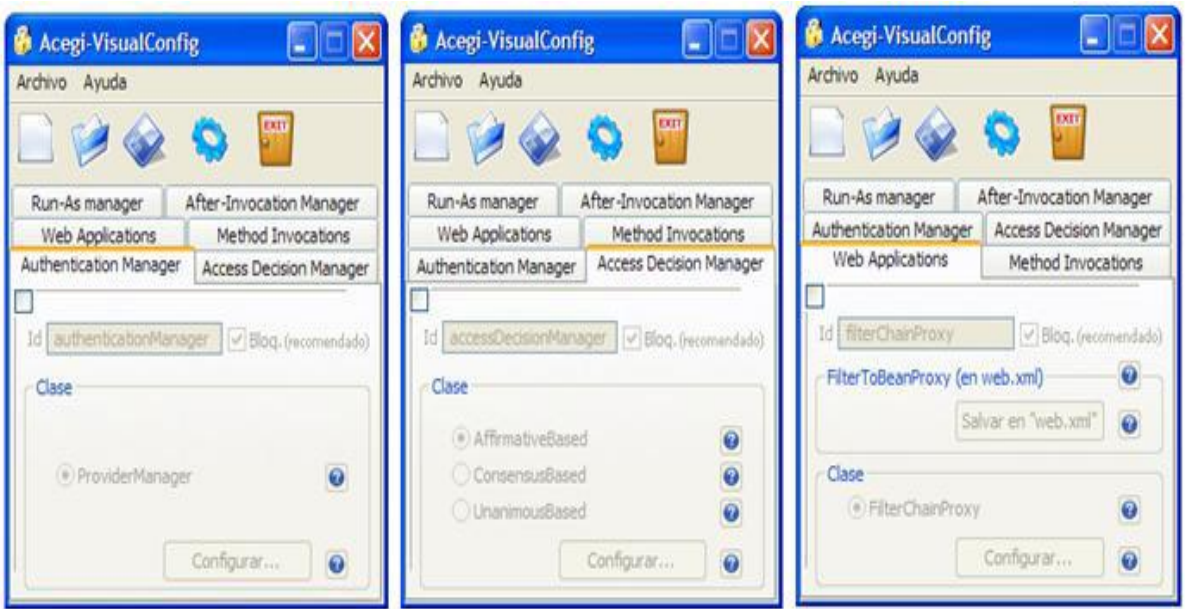

Figura 6: Ventana principal (*Authentication Manager*, *Access Decision Manager*, *Web Applications*).

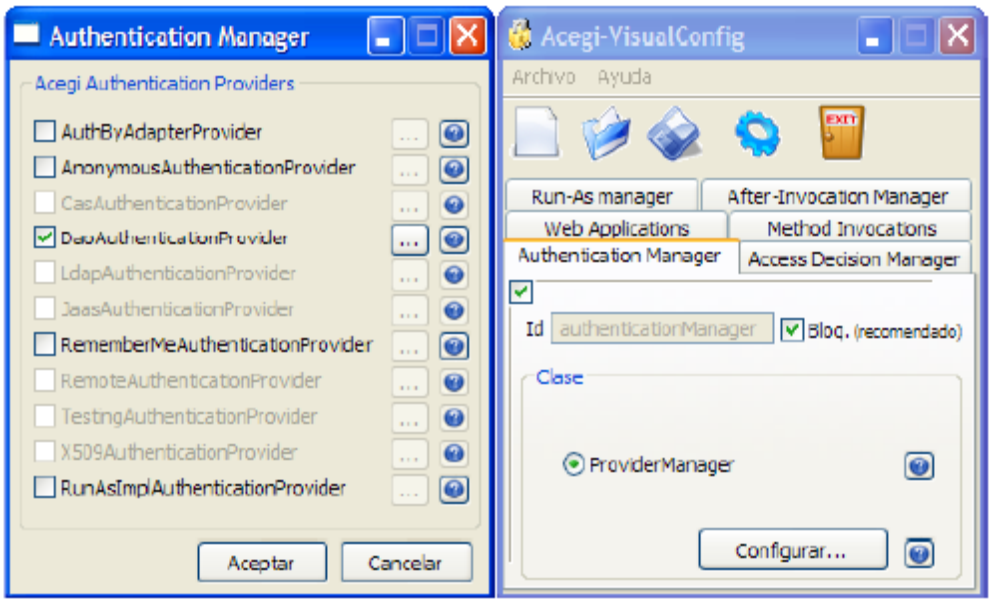

Figura 7: Ventana *Authentication Manager* (seleccionar *providers*).

Análisis de los resultados.

Para la validación de la propuesta de diseño de la aplicación descrita, son aplicadas un grupo de métricas técnicas, especialmente enfocadas en la evaluación de sistemas orientados a objetos, centrándose fundamentalmente en la medición de su calidad. Además, se evalúa el resultado obtenido al concluir la implementación de cada historia de usuario, a partir de los casos de prueba de aceptación propuestas por XP, comprobando así que las necesidades del cliente fueron satisfechas.

Métricas utilizadas.

Métricas CK.- Conjunto de métricas de productos específica para código orientado a objetos propuestas por Chidamber y Kemerer en su publicación "*A Metrics Suite for Object Oriented Design*", de 1994[12]. Tratan de medir la complejidad, acoplamiento, cohesión, herencia y comunicación inter-clases. Por otro lado no se establece ningún método de evaluación de calidad de producto ni se relacionan explícitamente los atributos de calidad con las propiedades del paradigma orientado a objetos, de forma intuitiva, eso sí, se sabe que el control de dichas propiedades ayuda, de manera general, a mejorar el diseño o el producto final[13].

- Profundidad del árbol de herencia (DIT).- La métrica DIT de una clase C es su profundidad en el árbol de herencia[13], a medida que el DIT crece, las clases de los niveles más bajos heredan mayor cantidad de métodos. Esto trae consigo potenciales dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (DIT largo) también conduce a una complejidad de diseño mayor, pero aporta como punto positivo la reutilización de un gran número de métodos[14]. DIT es solamente aplicable a las clases hijas (que heredan) del sistema y proporciona una idea sobre la complejidad de la herencia, en el diseño que

se evalúa. El mayor valor DIT obtenido es 3, lo que representa una jerarquía de clases poco profunda y por tanto una baja complejidad del diseño, facilitando el mantenimiento y la aplicación de pruebas a la herramienta.

- Número de descendientes (NOC).- El NOC de una clase es el número de subclases que están inmediatamente subordinadas a ella en la jerarquía[13]. Valores grandes de NOC representan un incremento en la reutilización y que la abstracción representada por la clase predecesora puede diluirse. Esto significa que existe la posibilidad de que algunos descendientes no sean miembros realmente apropiados de la clase predecesora. Esta métrica es aplicable solamente a clases que sirven de base a otras en el proceso de herencia. Como resultado de aplicar la métrica NOC a la herramienta, se tiene que el su máximo nivel es de 4 y corresponde a la clase nodoEtiquetado, valor que demuestra la existencia de un buen diseño de clases y una jerarquía bien estructurada.

- Falta de cohesión en los métodos (LOCM).- LOCM es el número de métodos que accede a uno o varios de los mismos atributos. Si no existen métodos que accedan a los mismos atributos, entonces LOCM = 0. En general, los valores altos para LOCM implican que la clase debe rediseñarse, descomponiéndola en dos o más clases[14]. Esta métrica se le aplicó a dos de las principales clases del sistema, la controladora principal *xmlMainClass*, y a *parserOut*, una de las clases de acceso a datos, específicamente la encargada de persistir la información de los objetos en los ficheros XML. Los resultados obtenidos de 3 y 4 respectivamente representan un nivel medio para los umbrales o medidas que proponen algunos autores en el campo de la métrica y el diseño, lo que determina que el nivel de diseño para estas clases es bueno.

Métricas LK.- Conjunto de métricas propuestas por Lorenz y Kidd en su libro "*Object-Oriented Software Metrics*", publicado el 29 de junio de 1994[15].

- Tamaño de clase (TC).- El tamaño general de una clase puede medirse a partir del total de operaciones (incluyendo las heredadas), y del número de atributos (incluyendo los heredados), encapsulados por la clase. Valores grandes de TC representan que la clase tiene una gran responsabilidad. Esto implica la reducción de su reutilización, complicando además la implementación y las pruebas. De forma general, operaciones y atributos deben ser ponderados al determinar el tamaño de la clase. Valores pequeños de TC representan clases más reutilizables [14]. Se le aplicó la métrica TC a un número de 14 clases para un total de 35 atributos, promediando 2.5, y 54 operaciones para una media de 3.85. Un total de 13 de las clases analizadas tienen tamaño pequeño, 1 tamaño medio y 0 tamaño grande. Más del 92% de las clases son clasificadas como pequeñas, respaldando esto de manera positiva el diseño del sistema, según los parámetros de calidad propuestos para esta métrica.

- Número de operaciones redefinidas para una sub-clase (NOI).- Son el número de operaciones heredadas de una superclase que son redefinidos por la subclase. Valores grandes de NOI, generalmente indican problemas en el diseño, provocando debilidad jerárquica de clases, y un *software* orientado a objetos que puede ser difícil de probar y modificar[14]. Como resultado de aplicar esta métrica al sistema (cuyo diseño está compuesto por 15 clases, 5 de las cuales son subclases y redefinen funciones heredadas), se obtuvo que de manera general existe en él una jerarquía

adecuada, lo que permite probar o modificar rápida y fácilmente la estructura del *software*.

- Índice de Especialización (IE).- El índice de especialización proporciona una indicación aproximada del grado de especialización de cada una de las subclases existentes en un sistema orientado a objetos. La especialización se puede alcanzar añadiendo o borrando operaciones, o bien por invalidación.

  IE = (NOI x nivel)/Mtotal

  En donde "nivel" es el nivel de la jerarquía de clases en que reside la clase, y Mtotal es el número total de métodos para la clase. Cuanto más elevado sea el valor de IE es más probable que la jerarquía de clases tenga clases que no se ajustan a la abstracción de la superclase.

Pruebas de aceptación.

En este punto la metodología propone un caso de prueba de aceptación por cada una de las historias de usuario. En cada una de estas pruebas se especifican un grupo de parámetros, como son las condiciones de ejecución, la entrada o pasos de ejecución y el resultado esperado, entre otros. En el caso de esta aplicación, la similitud de las historias de usuario (en cuanto a cómo debe implementarse cada una), hace que la mayoría de las pruebas compartan los valores de muchos de estos parámetros.

Conclusiones.

A partir de la investigación y el estudio del funcionamiento del *framework Acegi* se lograron determinar las características y requerimientos necesarios para la construcción de la herramienta propuesta.

El resultado, al culminar la implementación de la herramienta propuesta, es un producto que permite la edición y administración de las reglas y políticas de seguridad proporcionadas por *Acegi*, de manera sencilla e intuitiva. El diseño eficiente y poco complejo de la herramienta ofrece la posibilidad de adicionar nuevas funcionalidades o características, en caso de ser requeridas, permitiendo además el fácil mantenimiento del producto.

La concepción de un entorno visual intuitivo y amigable, brinda a los usuarios de la herramienta una fácil comprensión de la misma, aportando con esto facilidad de uso y rápida adaptabilidad.

Proyectivamente, se debe continuar y mejorar el desarrollo de la aplicación, incluyéndole la posibilidad de manejar o editar otras funcionalidades aportadas por el *framework Acegi*, que no fueron incluidas en esta primera versión. Además, proporcionarle a la aplicación la característica de determinar si la configuración general del XML no está completa, es incorrecta o si debe o puede mejorarse. La configuración general del XML no es más que el conjunto de configuraciones independientes de las partes necesarias, en todo el proceso de seguridad, como la autenticación, la decisión de acceso, los filtros, etc. El estado actual de la aplicación sólo permite la correcta edición de cada una de estas partes por separado, y de los elementos que las componen. Esto deja la relación de funcionamiento entre las partes al conocimiento de quien configura

el XML, posibilitando la omisión de alguna de ellas o su incorrecta utilización. También se debe ampliar el alcance de la aplicación a otras funcionalidades útiles del *framework Spring*, no relacionadas con la seguridad de aplicaciones, y que tienen formas similares de configuración.

Bibliografía.

[1] s.a. JavaHispano%20Acegi.pdf. 2008 [En línea]. Disponible en: oness.sourceforge.net. Consultado Enero 3, 2009.

[2]Microsystems S. 2008 [En línea] Acerca de la tecnología Java. Disponible en: URL www.java.com/es/about. Consultado Enero 6, 2009.

[3] Allamaraju S. Programación Java Server con J2EE. EE. UU.: A. Multimedia, Edición 1.3., 2002.

[4] Johnson R. Expert One-on-One J2EE Design and Development. EE. UU. : Wrox Press, 2003.

[5] Walls C. Spring in Action. EE. UU. : M.P. & Co., 2005.

[6] Harrop R, Machacek J. Pro Spring. EE. UU. : Apress, 2005.

[7] s.a.[En línea] 2008. acegisecurity.sourceforge.net. Consultado Enero 6, 2009.

[8] s.a. [En línea] 2008. Sprigframework.pdf. URL: www.springframework.org. Consultado Enero 10, 2009.

[9] s.a. [En línea] 2007. articles/acegisecurity/part1.jsp?source=archives. Disponible en URL: www.javalobby.org. Consultado Enero 8, 2009.

[10]s.a. [En línea] 2007. introduccion-a-acegi.pdf. Disponible en: tecnoblog.entel.es/wp-content/uploads/2007/05. Consultado Diciembre 11, 2008.

[11] Metodologias RUP y XP - [PROCESOS DE DESAROLLO]. [En línea] Sábado 5 de Mayo de 2007. Disponible en URL: http://jackopc.blogspot.com. Consultado Diciembre 12, 2008.

[12] Chidamber & Kemerer [En línea] 1994. A Metrics Suite for Object Oriented Design. Disponible en URL: http://intranet.uci.cu/biblioteca/ingenieriaygestiondesoftwarey/ametricssuiteforobjectorienteddesign.pdf. Consultado Noviembre 11, 2008.

[13] Arregui J, Olmedilla J. Revisión Sistemática de Métricas de Diseño Orientado a Objetos. Madrid: Universidad Politécnica de Madrid, Facultad de Informática, septiembre de 2005.

[14] Pressman RS. Ingenería de Software. Un enfoque práctico. Vol. I. Madrid: Universidad Politécnica de Madrid, Facultad de Informática, 1998.

[15] Lorenz, M, Kidd J. Object-Oriented Software Metric. S.e; s.l. 1994 (impresión ligera).

Husted T. Struts in Action. s.l. : Co., M.P., 2003.

Man KD. Java Server Faces in Action. s.l. : M.P. Co., 2005.

Bauer C. Hibernate in Action. s.l. : M.P. Co, 2005.

A propósito de programación extrema XP (eXtreme Programming). [En línea] 2006. Disponible en URL: http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml. Consultado Abril 11 2007.

BloodshedSoftware [En línea] 2007. Disponible en URL: http://www.bloodshed.net/devcpp.html. Consultado Abril 12 2007.

Ejemplo de desarrollo software utilizando la metodología XP. [En línea] 2005. Disponible en URL: http://www.dsic.upv.es/asignaturas/facultad/lsi/ejemploxp. Consultado Abril 9 2007.

javahispano.net/frs/shownotes.php?release_id=86. 93 26. [En línea] 2007. Disponible en URL: www.javaconganas.com/space/path/frameworks/spring/ioc. Consultado Mayo 10 2008.
introduccion_spring_framework_v1.0.pdf.[En línea]2006. Disponible en URL: www.javahispano.org/contenidos.item.action?id=3167000&menuId=NEWS. Consultado Mayo 10 2008.
tecnicas-de-programacion-inversion-de-control. [En línea] 2007. Disponible en URL: www.programacion.net/java/articulo/jap_injection. Consultado Mayo 5 2008.
S.a. Framework. Indianapolis: Wiley Publishing, Inc., 2005.
González CS . ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras. [En línea] 2004. Disponible en URL: http://oness.sourceforge.net/proyecto/html/index.html. Consultado Mayo 9 2008.
Johnson R. Introduction to the Spring Framework. [En línea] Mayo de 2005. Disponible en URL: http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework. Consultado Junio 5 2008.
Johnson R, Hoeller F et al. Professional Java Development with the Spring Extreme Programming: A gentle introduction. [En línea] 2005. Disponible en URL: http://www.extremeprogramming.org. Consultado Mayo 5 2008.
S.a. Spring Security. [En línea] 2005. Disponible en URL: http://www.acegisecurity.org/. Consultado Mayo 9 2008.
S.a. Spring Framework. [En línea] 2005. Disponible en URL: http://www.springframework.org. Consultado Mayo 9 2008.
S.a. Spring Source. [En línea] 2005. Disponible en URL: http://www.springsource.com. Consultado Mayo 9 2008.
Walls C, Breydenbach R. Spring in Action second edition. s.l. : M.P. Co., 2008.
Yale University. CAS (Central Authentication Service). [En línea] (s.f.). Disponible en URL: http://www.yale.edu/tp/auth. Consultado Mayo 5 2008.