Universidad de las Ciencias Informáticas

Facultad 3

Desarrollo de una herramienta para la administración de la seguridad en aplicaciones empresariales, basada en el framework Acegi

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

Autores: Fidel Alejandro Ortega Orihuela
Edier Campo Pupo

Tutor: Ing. Daynier Ruiz Rodríguez

Junio del 2008

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Fidel Alejandro Ortega Orihuela.	Edier Campo Pupo	Daynier Ruiz Rodríguez
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Daynier Ruiz Rodríguez (druiz@uci.cu)

Graduado de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas “UCI” de La Habana, en julio del 2007. Se ha desempeñando como profesor de la asignatura Programación III en la Facultad 3, ocupa actualmente el rol de arquitecto de software en el proyecto Convenio Cuba – Venezuela.

AGRADECIMIENTOS

...de Fidel Alejandro:

A mis padres, por todo lo que me enseñaron, por su aliento en los momentos más difíciles de la tesis y hacerme saber que yo sí podía, por indicarme siempre cuál era el camino a seguir y ser los principales artífices de que yo haya llegado hasta aquí.

A mi tutor, por proponer un tema de tesis tan interesante y darnos las primeras ideas para llevarlo a cabo.

A Jackie, por estar siempre conmigo en el mismo barco, por la ayuda gigantesca que me brindó para que la tesis pudiera ser redactada, por estar conmigo hasta altas horas de la madrugada mientras trabajaba, por calmarme cuando yo pensaba que las cosas no salían, por tener siempre un beso enorme para aliviar mis preocupaciones y sobre todo, por ese amor tan grande que me hizo llegar y me dio fuerzas para terminar algo que parecía imposible por el poco tiempo disponible.

A aquellos profesores que a lo largo de mi vida como estudiante me han enseñado cosas útiles e interesantes y despertado mi motivación por el maravilloso mundo del saber.

A todos aquellos que se interesaron por cómo iba mi trabajo de diploma y me dieron valiosas sugerencias y recomendaciones para llevar a feliz término esta tarea.

***Fidel Alejandro Ortega Orihuela**.*

...de Edier:

A mi madre y mi hermano, por alentarme y apoyarme a cada paso, por estar siempre conmigo y ser la fuerza que me ayuda a seguir.

A mi tía Elvia y a Pinpa, por estar siempre, incluso cuando por cosas de la vida les resulta imposible.

A Ikseidel e Ikyalis, por ser especiales.

A Chichi y Tere, por todo el apoyo y la ayuda que me brindaron desde el principio, por ofrecerme un hogar y el cariño de una familia durante estos seis años.

A toda mi familia, por apoyarme y quererme.

A mis primos de Stgo. de las Vegas, por toda la ayuda que me dieron.

A Marien y Yasmany (Nino), por la ayuda constante, por alentarme a seguir (a empujones muchas veces), por estar siempre dispuestos a escucharme, por todo el cariño y la amistad que nunca han dejado de brindarme, por ser sencillamente los mejores.

A Kiri y Jorgito, a sus padres, a mi sobri, ustedes son también parte de mi familia.

A mi "más mejor" amigo..., Augusto, por compartir conmigo cuatro de estos seis años, por apoyarme y ayudarme constantemente, por los muy buenos ratos que hemos pasado juntos, y por los malos también, eso es parte de la amistad, estar siempre, en las buenas y en las malas. Para mi eres y serás siempre más que un amigo, un hermano.

A Daynier mi tutor, por ayudarnos y guiarnos.

A mis maestros y profesores.

A todos mis amigos.

Y en especial Marien..., a tu Haier, y las meriendas "UCI" que siempre me dejabas para las largas madrugadas, sin eso habría sido imposible.

***Edier Campo Pupo**.*

DEDICATORIA

…de Fidel Alejandro:

A mi mamá en primer lugar, porque a ella debo la existencia y la razón, por darme un amor que no conoce límites, enseñarme lo bueno y lo malo de la vida con una dulzura inigualable, por tomarme de la mano cuando niño y caminar siempre a mi lado hasta que pude hacerlo solo, por disfrutar mis victorias como nadie, cada vez que tengo éxito pienso en ella y en su reacción cuando se entere, por hacerme su querubín y adorarme como su "niño", a pesar de que hace mucho que soy un hombre.

A mi papá, por estar siempre cerca y conmigo a de pesar de estar lejos unos cuantos años, por señalarme mis errores con firmeza y energía cuando nadie se atrevía hacerlo, por ser el mayor responsable de la formación de mi carácter, por sacarme de tantas situaciones difíciles y tirarme un millón de "cabos", por su madurez y sus consejos a la hora de tomar decisiones importantes, por enseñarme a ser valiente y a enfrentar los problemas en el momento justo, por su cariño y por todo lo anterior creo que no ha habido ni habrá un padre tan completo ni mejor que él.

A mi abuelo José Ramón, mi otro padre, por darme el cariño que sólo se le da un hijo, por defenderme aún a riesgo de buscarse problemas por mí, por enseñarme a ser hombre, por compartir conmigo tantos momentos buenos y estar siempre a mi lado, por despedirse de mí antes de partir, hubiera dado lo que no tengo por que estuvieras aquí físicamente en este momento, pero sé que desde donde te encuentras sonríes complacido, que yo llegara hasta aquí era al igual que mío, tu sueño también, siempre estarás en mi corazón.

A mi abuela Mireya, por sus sacrificios más grandes que el sol a lo largo de estos 23 años, por quererme a mí, su mejor y único nieto, como el hijo que nunca pudo tener, aunque ella no lo diga sé que ha sido así.

A Jackie, tú que haces de mi vida una eterna fiesta, por acariciarme con tu sonrisa, llenar mi vida con un amor que nunca antes me había hecho ser tan feliz como hasta ahora y hacerme sentir más contento que una cucaracha en una lata de azúcar cuando estoy a tu lado.

A los que se consideran mis amigos y me han demostrado siempre, sin fallar, que lo son en cualquier circunstancia.

A todas aquellas personas que sienten como suyos mis triunfos en cualquier esfera de la vida y sufren mis derrotas tanto como yo.

Fidel Alejandro Ortega Orihuela.

...de Edier:

A mi abuelita Onelia, por hacer de mí un hombre, por quererme y cuidarme siempre.

A mi abuelo Felix, porque gracias a ti estoy aquí hoy.

A mi mamá, por quererme y apoyarme siempre.

A mi hermano, por tu apoyo y tu cariño.

A mi tía Elvia, a Pinpa, al Negro y a Ikyalis.

A mi tía Elda, como me gustaría que estuvieses aquí hoy.

A mi abuelita Orfelina, que se cuanto me quieres.

A toda mi familia.

A mis amigos, los de verdad.

Para ti Marien, que se cuanto deseabas ver llegar a mi vida este momento.

Para todos ustedes, dedico este, mi máximo logro en la vida, porque ustedes son la razón de mí existir, porque sin ustedes hoy no sería quien soy.

Y para ti, que tanto te quiero...

Edier Campo Pupo.

RESUMEN

La seguridad en aplicaciones que se desarrollan sobre la plataforma Java Enterprise Edition (JEE), frecuentemente se administra 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 importante desventaja, toda la configuración de sus reglas y políticas de seguridad para una aplicación, se hace de forma manual en ficheros XML. El objetivo del presente trabajo es el desarrollo de una aplicación que permita administrar de forma fácil y eficiente estas configuraciones. Una herramienta que automatice este proceso aportaría un ahorro considerable de tiempo en los proyectos productivos de la facultad 3 y la UCI, además de la disminución casi a cero de la posibilidad de introducir errores de forma involuntaria al configurar estas reglas. Se empleó XP como metodología de desarrollo lo que permitió obtener un producto sencillo, intuitivo y fácil de utilizar en poco tiempo.

PALABRAS CLAVE
Seguridad de aplicaciones, Java Enterprise Edition, Framework Spring, Framework Acegi, Configuración de la seguridad.

Tabla de contenido

INTRODUCCIÓN

En la actualidad el vertiginoso desarrollo de la tecnología ha contribuido en favor de que exista un mayor intercambio de información a nivel mundial. Este intercambio es realizado a través de sistemas de información cada vez más complejos y necesarios en el desarrollo de cualquier entidad. Para suplir las necesidades de gestión de toda esta información se desarrollan diariamente por todo el mundo un gran número de aplicaciones informáticas con diversas herramientas y métodos de desarrollo.

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 (1).

Esta plataforma goza de una gran madurez tecnológica, es extremadamente eficaz, sorprendentemente versátil, y se ha convertido en un recurso inestimable que permite a los desarrolladores:

- La creación de software multiplataforma.
- Crear programas para que funcionen en un navegador web.
- Crear robustas aplicaciones para servidores.
- Desarrollar aplicaciones para dispositivos móviles.

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

La plataforma JEE es la edición empresarial de la plataforma Java y está especialmente pensada para la creación de aplicaciones web (2). 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:

- Struts, recomendado para aplicaciones de gran tamaño y complejidad, posee un conjunto de clases que usan los estándares JEE y ayudan a acelerar el desarrollo de aplicaciones web. Está basado en la arquitectura MVC, lo que permite garantizar la separación en capas, logrando aplicaciones escalables, reutilizables y con un alto nivel de profesionalidad (3).
- JavaServer Faces (JSF, o simplemente “Faces”), posibilita desarrollar aplicaciones web de una manera fácil, brindando poderosos componentes de interfaz de usuario. (4).
- Spring, centrado en manejar los objetos de negocio. Facilita el desarrollo de buenas prácticas de programación. Ayuda a resolver muchos problemas evitando el uso de los Enterprise JavaBeans (EJB) y provee una alternativa a los mismos que es apropiada para muchas aplicaciones. Spring puede usar Programación Orientada a Aspectos (AOP) para el manejo de transacciones y seguridad, sin tener que usar el contenedor de EJB (5).
- Acegi, especialmente pensado para la administración de la seguridad en aplicaciones y muy ligado al framework Spring. Es responsable de la autenticación de usuarios, el acceso a los recursos y el control de los permisos (5).
- Hibernate, trabaja sobre la capa de persistencia de cualquier aplicación. Realiza el mapeo entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java. El término utilizado es ORM y consiste en la técnica de realizar la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa. Constituye un motor de persistencia que implementa múltiples funcionalidades (6).

Aplicar una política de seguridad a una aplicación es un aspecto que afecta a prácticamente la totalidad de las aplicaciones empresariales, y si no se adopta desde una perspectiva correcta puede llegar a ser una carga que afectará y lastrará el desarrollo del sistema (7). Las políticas de seguridad concernientes a una aplicación con frecuencia cambian en el transcurso de su implementación, por lo que obliga a los programadores a estar actualizándolas constantemente. Por seguridad se entiende como tal la necesidad

de saber que el usuario es quien dice ser (autenticación), y permitirle acceso sólo a aquellos recursos necesarios (autorización) (7).

La plataforma JEE cuenta con varias opciones o mecanismos para manejar la seguridad en las aplicaciones, siendo el framework Acegi la más completa de ellas. Este framework 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. A partir de este inconveniente se genera como **situación problemática** el hecho de que en aplicaciones incluso de pequeño a mediano tamaño, estos 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.

Una vez mostrada la situación problemática y justificada la necesidad de su solución, se está en condiciones de formular el siguiente **problema científico** de la investigación**:**

¿Cómo 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?

Considerando como **objeto de estudio** la administración de la seguridad mediante el framework Acegi en aplicaciones desarrolladas en Java, y tomando como **campo de acción** el uso de Acegi como proveedor de seguridad para aplicaciones en desarrollo, en los proyectos productivos de la facultad 3 y en la Universidad de las Ciencias Informáticas, se definió el siguiente **objetivo general**:

Desarrollar una aplicación para administrar con facilidad y claridad las políticas de seguridad proporcionadas por Acegi en aplicaciones que se desarrollen con tecnología JEE y utilicen este framework.

Como vía para lograr el cumplimiento del objetivo general se definen un grupo de **objetivos específicos** de la investigación:

1. Realizar un estudio del estado del arte actual de las herramientas y métodos utilizados para la administración de la seguridad con Acegi tanto nacional como internacionalmente.

2. Analizar la necesidad de la utilización de herramientas para la administración de la seguridad con Acegi en la facultad 3 y en la Universidad de las Ciencias Informáticas y su impacto en el proceso de desarrollo de software donde se utilice este framework.
3. Realizar la implementación del sistema.

Las siguientes **tareas de la investigación** están dirigidas a dar cumplimiento a los objetivos específicos planteados:

1. Investigar sobre el funcionamiento del framework Acegi.
2. Desarrollar un estudio sobre la utilización del framework Acegi en los proyectos de la facultad 3 y la UCI, determinando las necesidades de uso y los aspectos particulares empleados en cada uno.
3. Desarrollar una aplicación que permita la fácil y clara administración de las políticas de seguridad proporcionadas por el framework Acegi para aplicaciones desarrolladas con JEE.

Una herramienta de administración de las reglas y políticas de seguridad proporcionadas por el framework Acegi para aplicaciones empresariales que se desarrollen con la plataforma JEE, es el importante **aporte práctico** que da como resultado la realización de las tareas de la investigación y el logro de los objetivos planteados. Con este fin fueron utilizados varios **métodos de la investigación**:

Teóricos:

- Analítico-sintético: Empleado para extraer los elementos más importantes relacionados con el objeto de estudio, centrándose en el análisis de manuales y documentos, que permitan elaborar conclusiones y determinar elementos que resulten prioritarios para el desarrollo de una herramienta que aporte una solución práctica al problema planteado.
- Histórico-lógico*:* Necesario para la valoración del estado del arte sobre el tema, habida cuenta que lo más avanzado de la teoría, a saber, la lógica del asunto, es siempre el resultado de la historia anterior de esa teoría.

Empíricos:

- Entrevistas*:* Como técnicas para la recopilación de información, en este caso con el objetivo específico de dar cumplimiento a la tarea de la investigación número dos. Se realizaron entrevistas a líderes y estudiantes de proyectos con conocimientos y experiencia en el tema.

Estructuración del contenido

Capítulo 1. Fundamentación del tema: Desarrollo de aplicaciones empresariales con la plataforma JEE y los frameworks Spring y Acegi, administración de la seguridad con Acegi. Sistemas o herramientas de administración de las reglas y políticas de seguridad proporcionadas por el framework Acegi. Metodología y herramientas utilizadas para el desarrollo de la solución propuesta.

Capítulo 2. Descripción del sistema: Diagrama y descripción de las clases del diseño. Artefactos generados por la metodología empleada, historias de usuario y tareas de ingeniería.

Capítulo 3. Análisis de los resultados: Pruebas al sistema, resultados obtenidos.

CAPÍTULO 1: FUNDAMENTACIÓN DEL TEMA

Este capítulo aborda aspectos del desarrollo de aplicaciones empresariales y la utilización de tecnologías Java con este fin. Primeramente son expuestos conceptos y características esenciales de los frameworks Spring y Acegi, haciendo mayor énfasis en la forma de uso de este último, sus ventajas y desventajas como proveedor de seguridad en aplicaciones. Es realizado un análisis del estado del arte de los sistemas de administración de las políticas y reglas de seguridad proporcionadas por Acegi, así como la necesidad de automatización de este proceso. Al final del capítulo se habla de manera breve, de la metodología y herramientas de desarrollo empleadas en el diseño e implementación de la solución propuesta.

1.1 Framework Spring

Spring es un framework de código abierto para el desarrollo de aplicaciones en la plataforma Java EE. La primera versión 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.
Beneficios arquitectónicos de Spring:

- Spring puede organizar eficazmente la capa intermedia de cualquier aplicación ya sea web o standalone (8).
- Facilita el desarrollo de buenas prácticas, reduce el costo de programación en una aplicación, la proliferación de Singletons y elimina la necesidad de usar distintos y variados tipos de ficheros de configuración (9). Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas, fáciles de usar, las cuales constituyen prácticas comunes en la industria.
- Las aplicaciones con Spring son fáciles de testear.
- Puede hacer el uso de la implementación de EJB una opción, siendo antes esta implementación determinante para la arquitectura de una aplicación. Ayuda a resolver problemas sin usar EJB, realizando el mismo tipo de funciones sin ellos y proporcionando una alternativa a los mismos que es útil en muchas aplicaciones (9). Esta es una de las principales ventajas que brinda Spring y una de las razones fundamentales por las cuales se ha popularizado en la comunidad de programadores. Spring también facilita la manipulación de los objetos, independientemente del uso o no de EJB.

- Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, facilitando el desarrollo de funcionalidades específicas y haciendo que la curva de aprendizaje sea favorable para el desarrollador. Es un framework hecho por programadores para programadores (9).
- Permite resolución de problemas típicos de aplicaciones empresariales de manera declarativa, estos servicios como manejo de transacciones, seguridad, entre otros, sólo eran brindados por servidores JEE.

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) (10).

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.
- Manejo de 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) (10). Otras funcionalidades que brinda este framework son las siguientes:

- Excelente integración con una selección de herramientas de acceso a datos, incluyendo JDBC, Hibernate, iBATIS y Java Data Objects (JDO). Cuando se utiliza el API de Spring para el acceso a datos utilizando cualquiera de estas herramientas se obtienen las ventajas del excelente manejo de transacciones del mismo.

- Un amplio conjunto de clases que permiten la creación de aplicaciones web. Además de proveer total integración con frameworks de desarrollo como Struts y JSF, Spring contiene su propio framework MVC, con un conjunto de tecnologías para las vistas como JSP, entre otros.
- Incluye un módulo que se encarga del flujo de las páginas en las aplicaciones web, que constituye un poderoso controlador para ser usado cuando las aplicaciones demandan complejos controles de navegación, y puede ser integrado con otras tecnologías web como Struts y JSF.
- La existencia de un módulo que abarca el soporte para la internacionalización de mensajes, servicios de empresariales como e-mail, acceso a repositorios JNDI, integración con EJBs, accesos remotos, entre otros (5) (8) (10).

Después de ver algunas características y ventajas que brinda el framework Spring, puede apreciarse que este facilita la construcción de aplicaciones Java. A diferencia de 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. De este último, se expondrán a continuación características y beneficios de su uso como proveedor de seguridad en el desarrollo de aplicaciones empresariales.

1.2 Framework Acegi

En las aplicaciones empresariales para lograr satisfacer las necesidades del cliente solo se debe dar cumplimiento a los requisitos funcionales, pero si además se tienen en cuenta los requisitos no funcionales el producto desarrollado tendrá una mayor calidad y aceptación. Dentro de estos requisitos no funcionales se destaca la seguridad, de la que vale señalar que su implementación en las aplicaciones empresariales es una tarea complicada y resulta muchas veces en código ligado con las funciones de negocio.

Acegi Security System (11) 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 (12). En el año que transcurre 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 (7).

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 (7). 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 (7).

Si bien existe el estándar 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 (7). 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 (13). La funcionalidad y soporte proporcionados por Acegi son mucho mayores, y además permite la integración con JAAS, utilizándolo en la fase de autenticación (7).

Acegi proporciona cuatro opciones principales de seguridad:

- Listas de control de acceso (ACL) web basadas en esquemas URL (14).
- Protección de métodos y clases Java utilizando AOP (14).
- Single Sign-On (SSO ACL) (14).
- Seguridad proporcionada por el contenedor Web (14).

Está formado por cinco componentes (figura 1.1):

- *Security Interceptor*: Previene el acceso a los recursos seguros de la aplicación. Delega el manejo de la seguridad a los restantes componentes.

- *Authentication Manager*: Es el encargado del 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*.

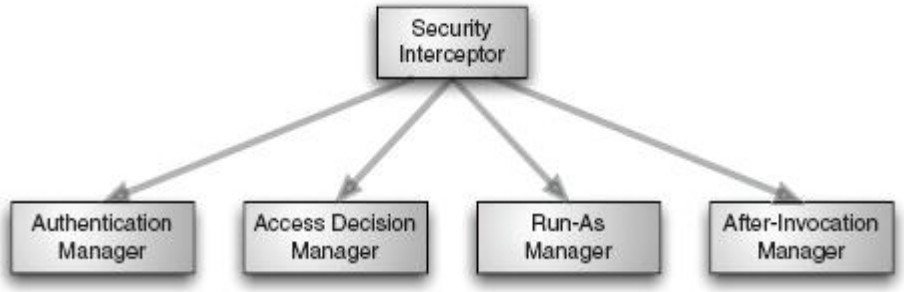


Figura 1.1 Componentes de Acegi.

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

- Autenticación, consistente en determinar la identidad del usuario, generalmente mediante un nombre de usuario y una contraseña.
- Control del acceso, consistente en determinar si un usuario previamente autenticado tiene acceso al recurso solicitado.

1.2.1 Autenticación

Antes de poder tomar decisiones sobre si un usuario puede acceder a un recurso, el usuario debe ser identificado para comprobar que es quien dice ser, tarea designada al interfaz *Authentication*, y desde el que se puede acceder a tres objetos:

- *Principal*, típicamente un nombre de usuario.
- *Credentials*, las credenciales del usuario que prueban que es quien dice ser, normalmente su contraseña, aunque podría ser otro tipo de información como certificados electrónicos.
- *Authorities*, una lista de los roles que posee el usuario o grupos a los que pertenece.

Cuando el usuario se autentica se crea un objeto *Authentication*, con los dos primeros objetos, principal y credenciales. En el caso de autenticación mediante nombre de usuario y contraseña se creará un objeto *UsernamePasswordAuthenticationToken* (7).

Acegi proporciona las clases necesarias para que esta autenticación se realice mediante usuario y contraseña, utilizando un adaptador para enlazar con la autenticación proporcionada por un contenedor de aplicaciones como son catalina, jboss, resin o jetty, o utilizando el servicio de Single Sign On que proporciona el proyecto CAS de la universidad de Yale (15).

Una vez creado este objeto *Authentication* se pasa al *AuthenticationManager*, que a partir del principal y las credenciales determina si éstas concuerdan con las esperadas, retornando el objeto *Authentication* tras añadirle las *authorities* correspondientes en caso afirmativo a través del método *authenticate()* o lanzando una excepción de tipo *AuthenticationException* en caso contrario (7). La interfaz *AuthenticationManager,* definida en *org.acegisecurity.AuthenticationManager* resulta sencilla de implementar como se muestra a continuación.

```
public interface AuthenticationManager {
      public Authentication authenticate(Authentication authentication)
      throws AuthenticationException;
}
```

Figura 1.2 Interfaz *AuthenticationManager.*

Acegi proporciona una implementación del gestor de autenticación *AuthenticationManager* que debería ser suficiente para la mayoría de los casos, *ProviderManager*. Esta clase tan sólo delega la autenticación en una lista de proveedores configurable, cada uno de los cuales implementa el interfaz *AuthenticationProvider* (7) (figura 1.3).

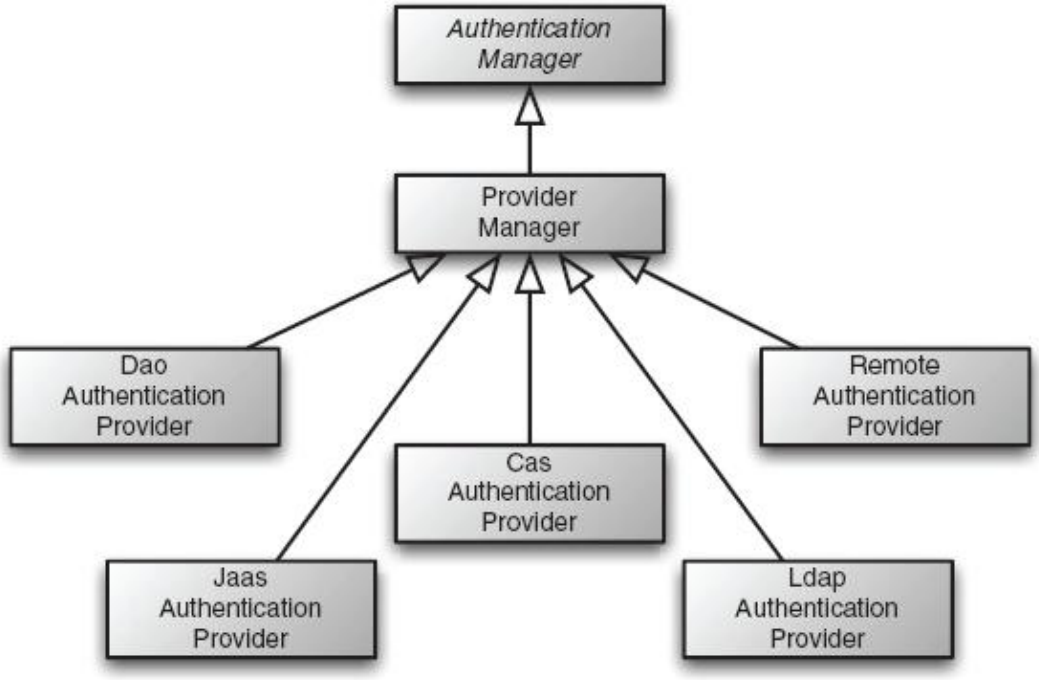


Figura 1.3 Algunos de los proveedores que implementan la interfaz *AuthenticationProvider*.

El objetivo del *ProviderManager* es delegar la autenticación de los usuarios a uno o varios proveedores de autenticación, en lugar de realizar esta tarea por sí mismo. *ProviderManager* va utilizando uno por uno todos los proveedores de autenticación listados y configurados hasta tanto el usuario sea autentificado con éxito o se llegue al final de la lista de proveedores. Un proveedor de autenticación está definido por la interfaz *org.acegisecurity.provider.AuthenticationProvider* (5).

ProviderManager puede configurarse en el fichero de configuración de Spring de la siguiente manera:

```
<bean id="authenticationManager" class="org.acegisecurity.providers.ProviderManager">
      <property name="providers">
            <list>
                  <ref bean="daoAuthenticationProvider"/>
                  <ref bean="ldapAuthenticationProvider"/>
            </list>
      </property>
</bean>
```

Figura 1.4 Configuración típica de *ProviderManager.*

Proveedor (`org.acegisecurity.*`)	Descripción
`adapters.AuthByAdapterProvider`	Se utiliza un adaptador para enlazar con la autenticación proporcionada por un contenedor de aplicaciones como por ejemplo: catalina, jboss, resin o jetty.
`providers.anonymous.` `AnonymousAuthenticationProvider`	Autentica al usuario como usuario anónimo. Útil cuando es necesario el token *user,* incluso cuando el usuario no se ha autenticado todavía.
`providers.cas.` `CasAuthenticationProvider`	Autenticación contra JA-SIG Central Authentication Service (CAS). Útil cuando se necesita capacidad de de Single Sign On.
`providers.dao.` `DaoAuthenticationProvider`	Recupera información de usuario, incluyendo el nombre de usuario y contraseña de una base de datos.
`providers.dao.` `LdapAuthenticationProvider`	Autenticación contra un servidor de Directorio de Acceso de Protocolo Ligero (LDAP).
`providers.jaas.` `JaasAuthenticationProvider`	Recupera información de usuario de un fichero de configuración de JAAS.
`providers.rememberme.` `RememberMeAuthenticationProvider`	Autentica a un usuario que se autenticó previamente y se recordó. Haciendo posible autenticar el usuario automáticamente sin necesidad de preguntar por su nombre de usuario y contraseña nuevamente.
`providers.rcp.` `RemoteAuthenticationProvider`	Provee autenticación contra un servicio remoto.
`providers.` `TestingAuthenticationProvider`	Para pruebas de unidad. Automáticamente considera un *Testing-AuthenticationToken* como válido. No debe ser usado en la producción.
`providers.x509.` `X509AuthenticationProvider`	Autenticación utilizando un certificado X.509. Útil para autentificar usuarios que son, de hecho, otras aplicaciones

	(como por ejemplo un servicio web).
`runas.RunAsImplAuthenticationProvider`	Autenticación de un usuario al cual se le ha sustituido su identidad por un *Run-As Manager*.

Tabla 1.1 Proveedores de autenticación proporcionados por Acegi.

Se puede pensar en el *AuthenticationProvider* como un subordinado del *AuthenticationManager* porque la interfaz *AuthenticationProvider* tiene un método *authenticate()* con la misma forma que el método *authenticate()* del AuthenticationManager (16).

El proveedor *DaoAuthenticationProvider* merece una atención especial. Este utiliza un objeto DAO para recuperar la información de usuario de una base de datos relacional. Con el nombre de usuario y la contraseña recuperados de la base de datos el *DaoAuthenticationProvider* realiza la autenticación comparándolos con el nombre de usuario y la contraseña pasados por el *ProviderManager* en el objeto *Authentication* (16). Realizar implementaciones de este interfaz es sumamente sencillo.

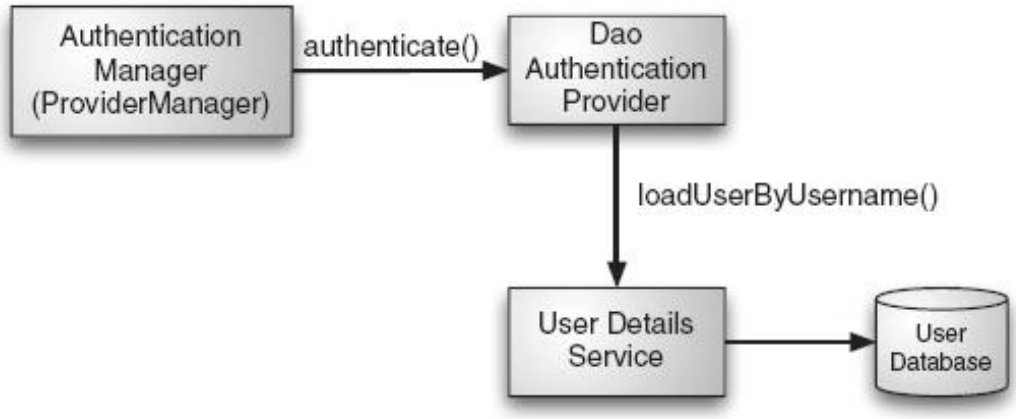


Figura 1.5. *DaoAuthenticationProvider* autentica a los usuarios en nombre del *AuthenticationManager* sacando la información del usuario de una base de datos.

La configuración de la seguridad se realiza en el fichero de configuración de Spring. Lo primero que debe ser declarado es el manejador de la autenticación, con la ruta y nombre de la clase que implementa la interfaz *AuthenticationManager* que será utilizada. Se le especifica además en la propiedad *providers* la lista de proveedores de autenticación que se desea utilizar. Para el caso en que se utilice la

implementación de la interfaz *AuthenticationManager* proporcionada por Acegi *(ProviderManager)* quedaría de forma igual o similar a, ver figura 1.4.

Cada proveedor de autenticación es configurado a su vez en el fichero de configuración de Spring de forma diferente, en dependencia de las características y necesidades particulares de cada uno de ellos. A continuación se muestra como configurar el proveedor *DaoAuthenticationProvider,* que es uno de los más utilizados.

```
<bean id="daoAuthenticationProvider"
   class="org.acegisecurity.providers.dao.DaoAuthenticationProvider">
      <property name="userDetailsService">
            <ref="authenticationDao"/>
      </property>
</bean>
```

Figura 1.6 Configuración típica de *DaoAuthenticationProvider.*

La propiedad *userDatailsService* es utilizada para identificar el *bean* que será empleado para recuperar la información de usuario de la base de datos. Esta propiedad espera una instancia de la interfaz *org.acegisecurity.userdetails.UserDetailsService* (16). Como *UserDetails* es una interfaz, se utiliza la clase *User* de Acegi para implementarla. Esta clase contiene el nombre de usuario, la contraseña, un bool que indica si el usuario está activo o no y un arreglo del tipo *GrantedAuthority*, clase del framework que representa los roles o permisos que contiene el usuario.

La interfaz *UserDatailsService* requiere de la implementación de solo un método *loadUserByUsername*:

```
public interface UserDetailsService {
      UserDetails loadUserByUsername(String username)
      throws UsernameNotFoundException, DataAccessException;
}
```

Figura 1.7 Interfaz *UserDatailsService*.

Acegi proporciona dos implementaciones de este interfaz, *InMemoryDaoImpl*, en la que la información de los usuarios se guarda en memoria, útil para la realización de pruebas, y *JdbcDaoImpl*, que accede a una base de datos a través de JDBC (5).

En el caso de *InMemoryDaoImpl* este configura la información que se quiere almacenar directamente en el fichero de configuración de Spring de la siguiente manera:

```
<bean id="authenticationDao"
   class="org.acegisecurity.providers.dao.memory.InMemoryDaoImpl">
      <property name="userMap">
            <value>
                  palmerd=4moreyears,ROLE_PRESIDENT
                  bauerj=ineedsleep,ROLE_FIELD_OPS,ROLE_DIRECTOR
                  myersn=traitor,disabled,ROLE_FIELD_OPS
            </value>
      </property>
</bean>
```

Figura 1.8 Ejemplo de configuración de *InMemoryDaoImpl.*

La propiedad *UserMap* es configurada con un objeto *org.acegisecurity.providers.dao.memory*.*UserMap* el cual define un conjunto de usuarios, contraseñas y privilegios. Afortunadamente no es necesario preocuparse por construir una instancia de *UserMap* cuando se utiliza *InMemoryDaoImpl,* porque existe un editor de propiedades que maneja la conversión de *String* a *UserMap* de forma automática y transparente. En cada línea del *userMap, String* es un par nombre-valor, donde nombre es el nombre de usuario, y valor una lista separada por comas que comienza con la contraseña del usuario y continúa con uno o más nombres que representan sus privilegios (16).

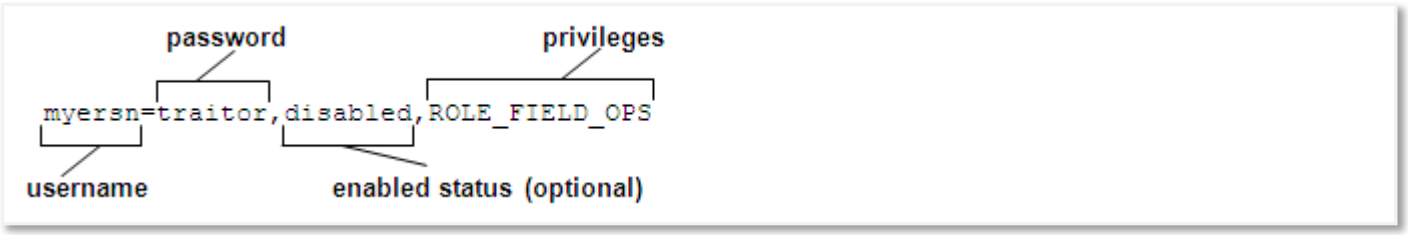


Figura 1.9 Ejemplo de un *String* de un *userMap*.

Las limitaciones de *InMemoryDaoImpl* son claras. En primer lugar al administrar la seguridad se requiere que se edite el fichero de configuración de Spring y se vuelva a desplegar la aplicación. Esto es aceptable y quizás aún útil en un entorno de desarrollo, pero es probablemente más dificultoso en la producción, no debiendo usarse en la misma (5).

La otra implementación de la interfaz *UserDetailsService* es *JdbcDaoImpl*, todavía flexible, en su forma más sencilla sólo necesita una referencia a la librería *javax.sql.DataSource* y puede ser configurada en el fichero de configuración de Spring (16).

```
<bean id="authenticationDao"
   class="org.acegisecurity.providers.dao.jdbc.JdbcDaoImpl">
      <property name="dataSource">
            <ref bean="dataSource"/>
      </property>
</bean>
```

Figura 1.10 Ejemplo de configuración de *JdbcDaoImpl.*

JdbcDaoImpl asume que existen ciertas tablas en la base de datos almacenando información de usuario. Específicamente, asume como ciertas las tablas *"Users"* y *"Authorities"* (16) mostradas en la figura 1.11.

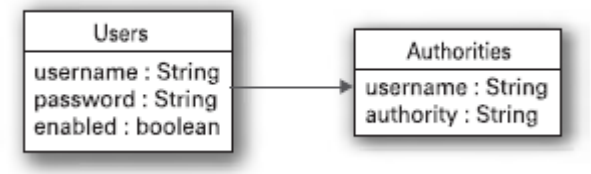


Figura 1.11 Las tablas de la base de datos asumidas por *JdbcDaoImpl.*

Las *querys* de SQL utilizadas por *JdbcDaoImpl* deben modificarse para que correspondan con las tablas de la base de datos que se va a utilizar.

1.2.3 Autorización

Una vez que el usuario está autenticado, entra en juego la parte del sistema encargada de la autorización, con el fin de permitirle acceso sólo a aquellos recursos para los que tiene privilegios. Para ello Acegi intercepta las llamadas a los objetos a través de la orientación a aspectos basada en AspectJ o proxies dinámicos, y las peticiones http, utilizando filtros. Así permite restringir tanto llamadas a métodos de determinadas clases o instancias como el acceso a URLs (7).

```
public interface AccessDecisionManager {
      public void decide(Authentication authentication, Object object,
         ConfigAttributeDefinition config)
         throws AccessDeniedException,
            InsufficientAuthenticationException;
      public boolean supports(ConfigAttribute attribute);
      public boolean supports(Class clazz);
}
```

Figura 1.12 Interfaz *AccessDecisionManager*.

Similar a la autenticación, para la autorización Acegi provee la interfaz manejadora de acceso *AccessDecisionManager,* figura 1.12.

El método *supports()* considera el tipo de clase del recurso seguro y estos atributos de configuración(los requerimientos de acceso del recurso seguro) determinan si el *AccessDecisionManager* es capaz de decidir el acceso al recurso seguro. El método *decide()* es donde esta última decisión es tomada. Si este método retorna un valor sin lanzar una excepción del tipo *AccessDeniedException* entonces se concede el acceso al recurso seguro (16).

Acegi proporciona tres implementaciones de *AccessDecisionManager* que se basan en el concepto de una votación, pero diferenciando las reglas de decisión:

Manejadores de acceso	Descripción
`org.acegisecurity.voteAffirmativeBased`	Permite el acceso si al menos uno de los votantes votó permitiendo el acceso.
`org.acegisecurity.vote.ConsensusBased`	Permite el acceso si el número de votos positivos permitiendo el acceso es mayor o igual que el de negativos.
`org.acegisecurity.vote.UnanimousBased`	Permite el acceso si todos los votantes votaron permitiendo el acceso.

Tabla 1.2 Implementaciones de *AccessDecisionManager* proporcionadas por Acegi.

Un *AccesDecisionVoter* es cualquier objeto que implementa la interfaz *org.acegisecurity.* vote.AccessDecisionVoter (16):

```
public interface AccessDecisionVoter {
      public static final int ACCESS_GRANTED = 1;
      public static final int ACCESS_ABSTAIN = 0;
      public static final int ACCESS_DENIED = -1;
      public boolean supports(ConfigAttribute attribute);
      public boolean supports(Class clazz);
      public int vote(Authentication authentication, Object object,
         ConfigAttributeDefinition config);
}
```

Figura 1.13 Interfaz *AccessDecisionVoter*.

Como puede apreciarse la interfaz *AccessDecisionVoter* es muy similar a la interfaz *AccessDecisionManager.* La diferencia es que en vez de tener un método *decide()* que no devuelve nada, aquí existe un método *vote()* que devuelve un entero. Esto se debe a que un objeto de tipo *AccessDecisionVoter* no decide si permitir o no el acceso (16), solamente vota en cuanto a tomar la decisión de si el acceso es concedido o no, como se muestra en la figura siguiente:

ACCESS_GRANTED	Vota porque se otorgue el acceso al recurso
ACCESS_DENIED	Vota porque se deniegue el acceso al recurso
ACCESS_ABSTAIN	Se abstiene, no puede decidir si se otorga o no el acceso

Figura 1.14 Formas de votar de los objetos de tipo *AccessDecisionVoter.*

AccessDecisionManager delega la facultad de emitir votos en objetos de tipo *AccessDecisionVoter*. El manejador de acceso se declara en el fichero de configuración de Spring, se le especifica en la propiedad *decisionVoters* los objetos votantes. El trabajo de estos objetos es analizar los permisos que tiene el usuario y los requeridos para acceder a un recurso, se puede implementar la interfaz *AccessDecisionVoter* o se puede utilizar la implementación *RoleVoter* que proporciona Acegi. Este objeto compara todos los atributos de configuración que presenta un recurso que comienza con el prefijo *ROLE_* con todos los permisos de acceso que presenta el usuario (contenidos en el arreglo de objetos *GrantedAuthority*), si alguno coincide autoriza el acceso de lo contrario lo deniega.

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.UnanimousBased">
      <property name="decisionVoters">
            <list>
                  <ref bean="roleVoter"/>
            </list>
      </property>
</bean>
```

Figura 1.15 Configuración típica de *AccessDecisionManager*.

El objeto *RoleVoter* solamente se abstendrá de votar cuando las *authorities* requeridas para el acceso no tengan el prefijo *ROLE_*. Por ejemplo si el recurso seguro sólo requiere *authorities* que no tengan dicho prefijo, como *CREATE_USER*, entonces el objeto *RoleVoter* se abstendrá (16). La clase *RoleVoter* se configura en el fichero de configuración de Spring de la siguiente manera:

```
<bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter"/>
```

Figura 1.16 Configuración típica de *RoleVoter*.

Aunque *ROLE_* es el prefijo por defecto, no es imposible utilizar otro. Esto se logra configurando la propiedad *rolePrefix* (16) (figura 1.17), ahora *RoleVoter* solamente dará votos de autorización a los privilegios que comiencen con el nuevo prefijo.

```
<bean id="roleVoter" class="org.acegisecurity.vote.RoleVoter">
      <property name="rolePrefix" value="GROUP_"/>
</bean>
```

Figura 1.17 Sustitución del prefijo por defecto *ROLE_*, por *GROUP_* en este caso.

Por defecto *AccessDecisionManager* deniega el acceso a un recurso si todos los votantes se abstienen. Sin embargo, este comportamiento puede cambiarse, modificando el valor de la propiedad *allowIfAllAbstain a true,* esta propiedad no es necesario configurarla si el valor que se le dará es *false,* su valor por defecto.

```
<bean id="accessDecisionManager" class="org.acegisecurity.vote.UnanimousBased">
      <property name="decisionVoters">
            <list>
                  <ref bean="roleVoter"/>
            </list>
      </property>
      <property name="allowIfAllAbstain" value="true"/>
</bean>
```

Figura 1.18 Configuración de la propiedad *allowIfAllAbstain* para que se conceda el acceso al recurso si todos los votantes se abstienen.

Si no se utiliza la clase *RoleVoter* es posible implementar la interfaz *AccesDecisionVoter* mediante *BasicAclEntryVote*r, que a su vez delega en una jerarquía de objetos que permite comprobar si el usuario supera las reglas establecidas, como listas de control de acceso (7).

RoleVoter es la opción más común, proporcionando una autenticación basada en grupos o roles, donde se permite el acceso si el usuario pertenece al menos a alguno de los configurados. En el segundo caso se permite restringir el acceso a objetos a nivel de instancia. En ambos casos el sistema que intercepta las llamadas debe ser configurado (7).

Los posibles recursos que se pueden proteger o impedir el acceso sin autorización son:

- URLs, mediante un filtro en el descriptor de aplicación web, *FilterSecurityInterceptor* (7).
- Métodos de objetos definidos en el contexto de aplicación de Spring, utilizando *MethodSecurityInterceptor* (7).
- Cualquier *PointCut* definible en AspectJ, mediante *AspectJSecurityInterceptor* (7).

Existen casos en los que la protección de las llamadas a métodos no es suficiente, necesitando protegerse de distinta forma distintas instancias de una clase. Como ejemplo se puede pensar en un sistema de ficheros, en los que cada archivo tiene distintos permisos, según si el usuario que accede a ellos es el dueño del archivo, pertenece al grupo del dueño o no cumple ninguna de las opciones anteriores.

Acegi proporciona en sus últimas versiones el soporte necesario para implementar seguridad basada en listas de control de acceso (7). Las clases clave que se deben conocer son:

- BasicAclEntryVoter, obtiene las listas de control de acceso del objeto llamado y vota sobre si se debe permitir el acceso a él o no (7).

- BasicAclAfterInvocationProvider permite denegar el acceso a un objeto después de que el método se haya invocado, útil cuando no se puede saber a priori (7).
- BasicAclAfterInvocationollectionFilteringProvider, similar al anterior, elimina los objetos a los que el acceso no ha sido permitido en aquellos métodos que devuelven colecciones (7).

1.2.4 Filtros

Si se quiere configurar la autenticación y el acceso a los recursos seguros en una aplicación web, estos procesos van a ser aplicados desde filtros. Los filtros son los encargados de interceptar las solicitudes a recursos web y aplicarles los procesos de seguridad, entre ellos pasarles las solicitudes al manejador de autenticación y de acceso.

Acegi provee de un conjunto de filtros que permiten implementar a cualquier nivel la política de seguridad, entre ellos se encuentran:

Filtros	Descripción
ChannelProcessingFilter	Es el encargado de analizar si la solicitud necesita ser entregada en canal seguro (HTTPS) o inseguro (HTTP), redireccionando para el canal adecuado.
Authentication-ProcessingFilter	Acepta peticiones de autenticación y las envía al manejador de autenticación para ser autenticadas.
CAS-processing filter	Acepta tickets del servicio de autenticación CAS como evidencia de que Yale CAS ha autenticado a un usuario.
HTTP Basic authorization filter	Procesa la autenticación usando el servicio de autenticación HTTP BASIC.
Integration filter	Maneja el almacenamiento de autenticación entre peticiones, por ejemplo, en HTTP Session.
Security enforcement filter	Asegura que un usuario ha sido autenticado y conoce los requerimientos de autorización para acceder a un recurso web restringido.
AuthenticationProcessingFilter EntryPoint	Es el encargado de impulsar al usuario a la autenticación, específicamente redireccionar al usuario a una página HTML con un formulario de login.

FilterSecurityInterceptor	Intercepta las solicitudes, utiliza el manejador de autenticación para determinar cuando un usuario está logueado y utiliza el manejador de acceso para determinar cuando un usuario tiene permiso para acceder a un recurso. En él se especifican los permisos necesarios para acceder a cada uno de los recursos web de la aplicación.

Tabla 1.3 Algunos filtros proporcionados por Acegi.

El contenedor de Spring y el uso de programación orientada a aspectos, permite realizar toda la configuración de estos filtros en el fichero de contexto, a diferencia de cuando se implementan filtros sin Spring que lleva a declararlos en el fichero web.xml de la aplicación.

La clase *FilterChainProxy* mapea cada uno de los filtros con el patrón de URL que estos interceptarán evitando declarar para cada uno las entradas *<filter-mapping>* y *<filter>* en el archivo *web.xml* de la aplicación. Para configurar todos los filtros se declara primeramente el *bean FilterChainProxy*, al cual se le especifica en el atributo *filterInvocationDefinitionSource* el patrón de URL y los filtros que interceptarán las solicitudes que cumplan con dichos patrones. El orden en que se coloquen los filtros significará el orden en que interceptarán la solicitud y luego se desencadenarán.

Normalmente cuando se trabaja con expresiones regulares para referenciar un determinado patrón de comportamiento se señala que previo de la comparación se convierta la expresión a minúscula para evitar fallar en comparaciones por diferencias entre minúsculas y mayúsculas, y además que se utilice el tipo de patrón de Apache Ant puesto que es el que resulta más familiar.

```
<bean id="filterChainProxy" class="org.acegisecurity.util.FilterChainProxy">
      <property name="filterInvocationDefinitionSource">
            <value>
                  CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                  PATTERN_TYPE_APACHE_ANT
                  /**=httpSessionIntegrationFilter,
                  ➥ authenticationProcessingFilter,
                  ➥ exceptionTranslationFilter,
                  ➥ filterSecurityInterceptor
            </value>
      </property>
</bean>
```

Figura 1.19 Configuración tipica de *FilterChainProxy.*

Seguidamente se escribe cada uno de los filtros a utilizar, en el orden correcto. En la declaración del *ChannelProcessingFilter*, se le especifica en la propiedad *filterInvocationDefinitionSource* cada una de las URL y el canal que requieren (seguro o inseguro), para ello se utilizan expresiones regulares.

```
<bean id="channelProcessingFilter" class="org.acegisecurity.securechannel.
   ➥ChannelProcessingFilter">
      <property name="filterInvocationDefinitionSource">
            <value>
                  CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                  PATTERN_TYPE_APACHE_ANT
                  /login.htm=REQUIRES_SECURE_CHANNEL
                  /j_acegi_security_check*=REQUIRES_SECURE_CHANNEL
                  /**=REQUIRES_INSECURE_CHANNEL
            </value>
      </property>
      <property name="channelDecisionManager" ref="channelDecisionManager"/>
</bean>
```

Figura 1.20 Configuración típica de *ChannelProcessingFilter.*

En la propiedad *channelDecisionManager* se referencia a un *bean* de la clase *ChannelDecisionManagerImpl*, esta clase de Acegi delega en los manejadores de canal (*SecureChannelProcessor* o *InsecureChannelProcessor*) para que examinen la solicitud y sus requerimientos de seguridad y de acuerdo a ello redireccionarla al canal adecuado.

```
<bean id="channelDecisionManager" class="org.acegisecurity.securechannel.
   ➥ ChannelDecisionManagerImpl">
      <property name="channelProcessors">
            <list>
                  <bean class="org.acegisecurity.securechannel.
                     ➥ SecureChannelProcessor"/>
                  <bean class="org.acegisecurity.securechannel.
                     ➥ InsecureChannelProcessor"/>
            </list>
      </property>
</bean>
```

Figura 1.21 Configuración típica de *ChannelDecisionManager.*

En el *AuthenticationProcessingFilter* se referencia en la propiedad *authenticationManager* al manejador de autenticación que se está utilizando, en la propiedad *authenticationFailureUrl* la URL a donde se redireccionará en caso de que falle la autenticación y en *filterProcessesUrl* la URL a la que se envía el formulario de autenticación.

```
<bean id="authenticationProcessingFilter" class="org.acegisecurity.ui.webapp.
   ➥AuthenticationProcessingFilter">
      <property name="filterProcessesUrl" value="/j_acegi_security_check"/>
      <property name="authenticationFailureUrl" value="/login.htm?login_error=1"/>
      <property name="defaultTargetUrl" value="/"/>
      <property name="authenticationManager" ref="authenticationManager"/>
</bean>
```

Figura 1.22 Configuración típica de *AuthenticationProcessingFilter.*

Cuando se declara el *AuthenticationProcessingFilterEntryPoint* se le especifica en la propiedad *loginFormUrl* la dirección de la página que contiene el formulario de autenticación y en la propiedad *forceHttps* se le da valor *true* para señalar que se use canal seguro cuando se trate la solicitud de autenticación.

```
<bean id="authenticationEntryPoint" class="org.acegisecurity.ui.webapp.
   ➥AuthenticationProcessingFilterEntryPoint">
      <property name="loginFormUrl" value="/login.htm"/>
      <property name="forceHttps" value="true"/>
</bean>
```

Figura 1.23 Configuración típica de *AuthenticationProcessingFilterEntryPoint.*

Por último en el *FilterSecurityInterceptor* se especifica en la propiedad *authenticationManager* el manejador de autenticación que se utiliza, en la *accessDecisionManager* se referencia el manejador de acceso y por último en la propiedad *objectDefinitionSource* se declara, auxiliándose de expresiones regulares, las URL de las solicitudes y los roles o permisos para acceder a estas.

```
<bean id="filterSecurityInterceptor" class="org.acegisecurity.intercept.web.
   ➥FilterSecurityInterceptor">
      <property name="authenticationManager" ref="authenticationManager"/>
      <property name="accessDecisionManager" ref="accessDecisionManager"/>
      <property name="objectDefinitionSource">
            <value>
                  CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                  PATTERN_TYPE_APACHE_ANT
                  /configuracion.xml**=ROLE_ACCESO_DENEGADO
                  /**=ROLE_ACCESO_DENEGADO
            </value>
      </property>
</bean>
```

Figura 1.24 Configuración típica de *FilterSecurityInterceptor.*

Para el caso de aplicaciones web existen tres formas de que un usuario se autentique:

- Utilizando autenticación de tipo BASIC, definida en el RFC 1945, el usuario introduce su usuario y contraseña en una simple ventana emergente del navegador. Es necesario en el caso de que se quieran añadir características de seguridad a servicios web.
- Autenticándose mediante un formulario web, es la forma más habitual ya que permite integrar el formulario de login en la aplicación web.
- Utilizando el servicio de autenticación central CAS de la Universidad de Yale, en caso de que se requieran características de Single Sign On, de forma que el usuario sólo tiene que autenticarse una vez para todos los servicios que puedan proporcionarse en el ámbito de una empresa, incluso en distintos servidores y desarrollados con distintos lenguajes de programación.

Cada una de las formas anteriores requiere de la configuración del filtro correspondiente en el descriptor de aplicación web, *BasicProcessingFilter*, *AuthenticationProcessingFilter* o *CasProcessingFilter* respectivamente. La forma de configurarlos es definir los filtros como del tipo *FilterToBeanProxy*, delegando, según un parámetro de inicialización, en uno de los filtros anteriores definidos en el contexto de aplicación de Spring, lugar donde pueden ser más fácilmente configurados (7).

Los clientes llamados “ricos” o aplicaciones standalone también están soportados, utilizando un gestor de autenticación remoto cuya implementación utiliza un servicio web en el lado del servidor, utilizando *RemoteAuthenticationManager* y *RemoteAuthenticationProvider* (7).

1.2.6 Ventajas del framework Acegi

En las aplicaciones web Acegi usa filtros que interceptan las solicitudes, utilizando un único mecanismo para declararlos e inyectarlos con sus dependencias utilizando IoC, creando un desacople total e independencia de la aplicación. Además permite implementar la seguridad a bajo nivel controlando las invocaciones de métodos, usando AOP los objetos proxies de Acegi aplican aspectos que aseguran que el usuario que llama a los métodos contenga los permisos requeridos (5).

Otras ventajas que brinda Acegi:

- Provee un paquete de listas de control de acceso con ventajas tales como enmascaramiento de los bits enteros, herencia de permisos, incluido el bloqueo, el estándar JDBC respaldado por un repositorio de listas de control de acceso, caché y un diseño manejado por interfaces conectadas (11).
- Permite mantener a los objetos de negocio libres de código de seguridad (11).
- Da la posibilidad que la autenticación sea manejada a través de la solución Siteminder de CA, lo cual es muy común en grandes entornos corporativos (11).
- Brinda de forma transparente, tan sólo estableciendo unos parámetros de configuración, soporte para cifrado de contraseñas (SHA y MD5), caché de la información de autenticación y redirección automática de peticiones http a canales seguros https para aquellas URLs que se deseen. También proporciona inusuales combinaciones de puertos (incluyendo si acceden mediante un servidor intermedio como Apache) y decisiones de transporte que pueden ser conectadas (11).
- Además de brindarle seguridad a los *beans* de la aplicación, Acegi también asegura las peticiones http, no siendo necesario confiar en las restricciones de seguridad de web.xml. Mejor todavía, las peticiones http pueden ser ahora aseguradas por las expresiones regulares escogidas o la utilización de patrones estilo Ant, junto con autenticación, autorización y reemplazo de permisos que pueden ser conectados (11).
- Soporta autenticación HTTP BASIC, esta autenticación es adecuada para aquellas aplicaciones que prefieren una simple ventana de login del navegador en lugar de un formulario de login (11).
- Para una mayor seguridad que la ofrecida por HTTP BASIC, Acegi también soporta Digest Authentication (el cual nunca envía la contraseña del usuario a través de la red). Digest Authentication es ampliamente soportado por los navegadores modernos (11).

- Puede leer con facilidad el certificado X509 del lado del cliente para la autenticación de los usuarios (11).
- Permite seleccionar el método a usar para configurar el entorno de seguridad, puede ser en el contexto de aplicación de Spring, a través de Jakarta Commons Attributes, mediante las características de anotación de Java Development Kit 5 o como algunos usuarios que extraen estos datos de configuración de una base de datos (11).
- Ofrece la posibilidad que el sistema entero de seguridad funcione dentro de una aplicación web usando los filtros proporcionados, sin necesidad de hacer cambios especiales o desplegar librerías al Servlet o el contenedor de EJB. Además la colección de credenciales y capacidades de autorización del servlet o el contenedor de EJB pueden ser utilizadas completamente a través de los llamados adaptadores de contenedores como Catalina (Tomcat), Jetty, JBoss and Resin, con contenedores adicionales que pueden ser fácilmente añadidos (11).
- Librería de etiquetas, proporciona una librería de etiquetas que puede ser utilizada en JSPs para garantizar que el contenido protegido como enlaces y mensajes sean únicamente mostrados a usuarios que posean los permisos adecuados. Esta librería de etiquetas se integra completamente con los servicios de las listas de control de acceso que proporciona Acegi, obteniendo información extra del logueado principal (11).
- Distintos métodos de almacenamiento de la información de autenticación. Incluye la posibilidad de obtener los usuarios y permisos utilizando ficheros XML, fuentes de datos JDBC o implementando un interfaz DAO para obtener la información de cualquier otro lugar (11).
- Permite la autenticación contra un directorio LDAP (11).
- Validación de cada dependencia de los objetos críticos y parámetros de configuración en el inicio del contexto de aplicación lo que garantiza que los errores de configuración en las políticas de seguridad sean detectados de forma temprana y corregidos rápidamente (11).
- Brinda soporte remoto a los proyectos que usan clientes ricos debido a que se integra con los protocolos remotos estándares del framework Spring porque procesa automáticamente los encabezados de autenticación HTTP BASIC que presentan dichos protocolos (11).
- Permite el acceso al objeto resultante de tipo *Authentication* a través del método *getRemoteUser()* y otros métodos de seguridad en HttpServletRequest (11).
- Además de proteger métodos que están siendo invocados en primer lugar, también puede tratar con los objetos retornados por los métodos. Incluye implementaciones que después de una

invocación hacen que los mecanismos de seguridad puedan lanzar una excepción o mutar el objeto devuelto basado en listas de control de acceso (11).

- Incluye pruebas de unidad, con una cobertura muy alta para dichas pruebas. Acegi provee un número de clases que brindan asistencia a las pruebas de unidad de los objetos seguros del negocio (11).
- Transfiere automáticamente el núcleo de la información de autenticación de una máquina a otra usando varios protocolos, incluyendo RMI y HttpInvoker del framework Spring (11).

1.2.7 Desventajas del framework Acegi

La configuración de las políticas de seguridad utilizando el framework Acegi se hace manualmente en un fichero XML. En la medida que las aplicaciones que están siendo desarrolladas crecen de tamaño la configuración de dichas políticas puede traer las siguientes las siguientes desventajas:

- Gasto grande de tiempo por parte de los desarrolladores encargados de implementar estas políticas de seguridad.
- Aumento sustancial de la probabilidad de cometer errores debido a lo engorroso y complicado que se va tornando la configuración de dichas políticas. Estos errores pueden desencadenar trastornos en la seguridad y desencadenar un mal funcionamiento del sistema.
- La revisión y actualización de estas políticas de seguridad se haría difícil y con poca claridad, trayendo demoras y dificultades en el desarrollo de las aplicaciones.
- Es un framework bastante complicado y difícil en un inicio, lo que pudiera impedir una rápida adaptación a él y causar desmotivación a los desarrolladores interesados en utilizarlo para manejar el tema de la seguridad, no cambiando entonces de otra tecnología al uso de Acegi debido a lo anterior.

1.3 Sistemas de administración de seguridad con Acegi

La existencia de algún tipo de herramienta o sistema que permita la administración de las políticas y reglas de seguridad proporcionadas por el framework Acegi, aportaría un importante punto de comparación y apoyo para el desarrollo de la aplicación propuesta por el presente trabajo de diploma. Con el objetivo de encontrar cualquier producto con estas características se realizó una amplia y exhaustiva búsqueda de información a través de Internet, en los proyectos productivos de la facultad 3 y en la UCI en

general. Solo fue encontrada una herramienta web, todavía en proceso de desarrollo, que permite automatizar de forma muy específica algunas de las funcionalidades que brinda el framework. La herramienta pertenece específicamente al proyecto productivo SIGEP de la facultad 4, y está estrechamente relacionada a características muy propias del mismo.

Partiendo de los resultados de esta investigación se determinó la gran importancia que tiene el desarrollo de la aplicación propuesta, dado que no se conoce de la existencia de ninguna herramienta similar que pueda ser utilizada en los proyectos de la facultad 3 y de la UCI.

1.7 Metodología utilizada

Todo desarrollo de software es riesgoso y difícil de controlar, y si no se emplea una metodología que guíe este proceso, los resultados obtenidos son clientes insatisfechos y desarrolladores aún más descontentos. Actualmente a nivel mundial, en dependencia del tiempo de vida y la complejidad del proyecto que se vaya a desarrollar se proponen diferentes metodologías.

Una metodología es el conjunto de técnicas y procedimientos que permiten conocer los elementos necesarios para definir un proyecto de software, es la base para la edificación de un producto de este tipo. Esencialmente, sirve para subir la "calidad" del software y controlar de manera transparente todo el proceso de desarrollo. Si se quiere que un proyecto sea escalable y flexible a los cambios es lógico pensar que para lograr ese propósito se necesite tomar en cuenta una de las muchas metodologías para el proceso de desarrollo de software que existen.

Las metodologías dadas sus características, se enmarcan en dos grandes grupos, los llamados "métodos pesados" y los "métodos ágiles". La diferencia más notable entre estos dos grupos es que mientras los métodos pesados intentan obtener los resultados apoyándose principalmente en la documentación ordenada, los métodos ágiles tienen como base de sus resultados la comunicación e interacción directa con todos los usuarios involucrados en el proceso.

La metodología utilizada para el desarrollo de la herramienta propuesta como solución al problema de la investigación fue XP (*Xtreme Programing*).

XP es una metodología ágil basada en cuatro principios: simplicidad, comunicación, retroalimentación y valor. Además está orientada por pruebas y refactorización, se diseñan e implementan las pruebas antes de programar la funcionalidad y el programador crea sus propios tests (pruebas) de unidad. (17)

Esta metodología se apoya en el trabajo orientado directamente al objetivo, basándose en las relaciones interpersonales, en la velocidad de reacción para la implementación y los cambios que puedan surgir

durante el desarrollo del proceso. Por ello se requiere mantener dentro del equipo a un representante "competente" del cliente, quién responderá a todas las preguntas y dudas que surjan del equipo de desarrollo durante el proceso, de forma que no se retrase la toma de decisiones. (18)

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. 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 siguientes iteraciones. XP genera solo cuatro artefactos (anexo I, II, III, IV), de los cuales solo tres son documentados; el último de ellos, las tarjetas CRC (anexo IV), son pequeñas tarjetas de cartón que se mantienen en poder del equipo de desarrollo mientras dura el proceso de construcción. (17) (18)

Una característica importante de XP es la producción siempre en parejas del código, las cuales van cambiando constantemente para lograr así que todo el equipo sepa y pueda modificar el código generado según las necesidades. Esto logra que los integrantes del equipo de desarrollo aprendan entre sí y compartan dicho código totalmente. En la actualidad XP se proyecta a ser un modelo de desarrollo común, sencillo y adaptable a las características cambiantes y exigentes de empresas y clientes. (18)

Es válido aclarar que las metodologías no son “camisas de fuerza” para los procesos de desarrollo de software sino guías para este, por lo que pueden ser adaptadas a las necesidades del equipo de desarrollo. XP solamente genera la documentación básica y absolutamente indispensable, por lo que se decidió además adoptar algunas características de otras metodologías de forma independiente para apoyar la descripción del sistema, como son el diagrama y la descripción de clases del diseño que se presentan en el capítulo dos.

1.8 IDE y lenguaje de programación

Un IDE es una herramienta de software que facilita el trabajo de los programadores, ofreciéndoles un gran número de funcionalidades en tiempo de implementación, como pueden ser componentes visuales, completamiento de código y editores de código especializados. Estas herramientas de software tienen asociados siempre uno o varios lenguajes de programación, los cuales permiten desarrollar productos informáticos para uno u otro entorno.

El IDE de desarrollo seleccionado fue el wxDev-C++ (anexo V) 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 haya conocido. C++ fue el primer lenguaje de programación impartido en la Universidad de las Ciencias Informáticas y el equipo de trabajo cuenta con un alto dominio del mismo.

1.9 Herramientas CASE

Las Herramientas CASE (*Computer Aided Software Engineering*) son aplicaciones informáticas destinadas a aumentar la productividad en el proceso de desarrollo de software reduciendo sus costes. Fue seleccionada para la creación del diagrama de clases del diseño la herramienta ArgoUML (anexo VI) por ser open-source y soportar el estándar UML 1.4.

Conclusiones

- El manejo de las políticas de seguridad constituye un aspecto muy importante en el desarrollo de aplicaciones empresariales.
- Existen varias maneras para configurar estas políticas en JEE, como el estándar JAAS y el framework Acegi.
- El framework Acegi, a pesar de sus beneficios presenta limitaciones a la hora de configurar las reglas y políticas de seguridad en las aplicaciones donde se utilice.
- Una herramienta que permitiera a los desarrolladores configurar las políticas de seguridad de las aplicaciones utilizando el framework Acegi permitiría:
  - Un ahorro considerable de tiempo al editar las políticas y reglas de seguridad.
  - Fácil y rápida configuración de las mismas, permitiendo que el cambio, actualización o adición de nuevas reglas y configuraciones no resulte un trabajo engorroso y demasiado complicado.
  - La disminución casi a cero de la posibilidad de cometer errores de configuración.
  - Los desarrolladores que recién comienzan a utilizar el framework Acegi podrían adaptarse rápida y fácilmente a su uso, eliminando el deseo de cambiar de tecnología sólo por lo complejo que puede parecer el trabajo con este framework en un principio.

CAPÍTULO 2: DESCRIPCIÓN DEL SISTEMA

El presente capítulo describe la arquitectura base de la aplicación, y expone desde una óptica general las características fundamentales del sistema mediante un diagrama de clases del diseño, además de la descripción del objetivo de cada una de las clases y sus responsabilidades. Se mencionan y explican brevemente tanto los atributos de calidad tenidos en cuenta para el diseño e implementación del producto como los patrones de diseño empleados, y el aporte o solución que estos brindan a la herramienta.
En respaldo a la documentación del software son elaborados los artefactos generados por la metodología seleccionada. También se ilustran los principales componentes de la interfaz gráfica del sistema.

2.1 Arquitectura de la aplicación

La arquitectura de software es un aspecto muy importante a tener en cuenta para el desarrollo de cualquier producto informático, pues esta define la estructura o forma del sistema en general, y en ella están comprendidas las propiedades visibles de los componentes del software y las relaciones entre ellos.
Esta herramienta 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.
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.
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 una total independencia al cumplir con sus responsabilidades, solo apoyándose entre ellas para desarrollar tareas cuyas actividades requieran ser procesadas por más de una capa.

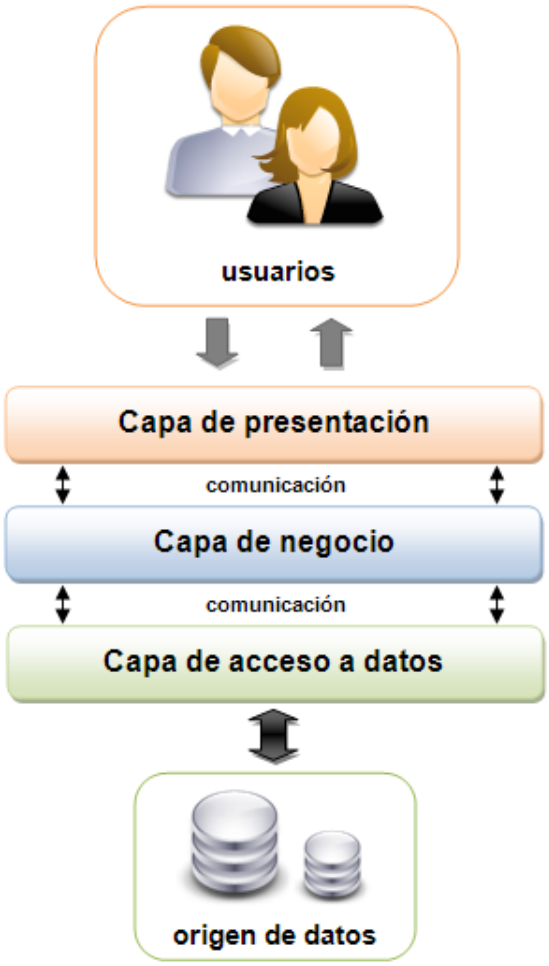


Figura 2.1 Separación lógica en capas.

A pesar de que el modelo de tres capas ha evolucionado a un modelo de n-capas incorporando capas intermedias que permiten desacoplar las primitivas y distribuirlas, esto sólo resulta beneficioso en sistemas de gran formato, por lo que en el caso de la aplicación objeto de este trabajo se utiliza este modelo en su forma primitiva, debido al pequeño formato de la misma.

Capa de presentación: Esta es la capa responsable de la interfaz de usuario, utiliza componentes visuales a través de los cuales se realiza todo el proceso de comunicación con el usuario. Es la encargada de mostrar los resultados de salida, y de recibir los datos de entrada, validando en ambos casos que la información posea el formato correcto.

Capa de negocio: En esta capa es donde se implementan todas aquellas funcionalidades que el sistema debe tener, ella se encarga de hacer cualquier tipo de operación solicitada por el usuario a través de la capa de presentación. Pasa los resultados obtenidos a cualquiera de las otras dos capas, para ser persistidos, mostrados a los usuarios o ambas inclusive. Controla además el acceso a los servicios del negocio y la invocación a la capa de acceso a datos.

Capa de acceso a datos: A esta capa le son delegadas las responsabilidades de recuperar o persistir entidades o cualquier tipo de información que el sistema necesite, es decir, es la responsable de leer o escribir la información manejada por las capas superiores en cualquier medio de almacenamiento de datos utilizado por el sistema, ya sea una base de datos o un fichero.

2.2 Atributos de calidad

Los modelos de arquitectura responden a una serie de aspectos que permiten su evaluación, posibilitando determinar si la arquitectura a implementar es realmente lo que se necesita. Son muchos los aspectos que se pueden tener en cuenta en este caso, pero es necesario centrar la atención solo en aquellos que realmente son importantes para el producto que se quiere desarrollar.

Dadas las características de esta aplicación los principales aspectos de calidad a tener en cuenta son:

- Funcionalidad: Habilidad del sistema para realizar el trabajo para el cual fue concebido (Kazman et al., 2001).
- Modificabilidad: Es la habilidad de realizar cambios futuros al sistema. (Bosch et al. 1999).
- Mantenibilidad: Es la capacidad de someter un sistema a reparaciones y evolución (Barbacci et al., 1995). Capacidad de modificar el sistema de manera rápida y a bajo costo (Bosch et al. 1999).
- Escalabilidad: Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2002).

La aplicación será evaluada en el capítulo tres teniendo en cuenta estos parámetros de calidad, apoyándose en la utilización de métricas especialmente concebidas para probar el diseño de sistemas orientados a objetos. Los resultados de estas pruebas darán una medida del cumplimiento de cada uno de estos atributos, permitiendo hacer cambios al diseño para mejorar la calidad del producto.

2.3 Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Sólo es utilizado un patrón de diseño en esta solución, el patrón Singleton (instancia única), diseñado para restringir la instanciación de objetos a partir de una clase a un único objeto, garantizando que sólo tenga una instancia y proporcionando un punto de acceso global a ella.

El patrón Singleton es utilizado 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).

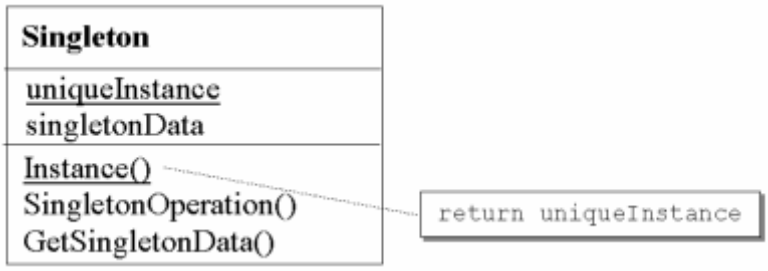


Figura 2.2 Estructura general del patrón Singleton.

2.4 Historias de usuario

En este epígrafe se presenta el artefacto más importante generado por XP, las “Historias de usuario” (anexo I), estas cumplen un objetivo muy similar al de los casos de uso en metodologías pesadas como RUP. Una historia de usuario es la descripción en lenguaje formal y sin tecnicismos de un requisito que debe cumplir el sistema a implementar. 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.

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: Edier Campo Pupo	
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:	

Tabla 2.1 Historia de usuario #1.

Historia de Usuario	
Número: 2	**Nombre historia:** Configurar *providers*
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: Edier Campo Pupo	
Descripción: Configurar los *providers* listados en el <*bean*> de la autenticación, debe permitir la configuración de los	

nodos <*bean*> para cada *provider* seleccionado, con todas las propiedades y requisitos necesarios, de forma fácil y sugerente. Hacer énfasis en los *providers* más utilizados y por tanto más demandados, como la autenticación contra bases de datos.
Observaciones:

Tabla 2.2 Historia de usuario #2.

Historia de Usuario	
Número: 3	**Nombre historia:** Configurar *daoAuthenticationProvider*
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: Edier Campo Pupo	
Descripción: Configurar *daoAuthenticationProvider*, configurar el <*bean*> *daoAuthenticationProvider*, debe permitir la configuración de todas las propiedades necesarias, de forma fácil y sugerente. En este caso ha de ser posible cambiar la configuración entre bases de datos y listas de usuarios en memoria de forma sencilla y rápida. Esto se debe a que al finalizar el proyecto la autenticación casi siempre se hará contra bases de datos, pero en el período de desarrollo del mismo las pruebas contra listas de usuarios en memoria son muy útiles y eficientes. También tienen que poderse configurar el uso de contraseñas encriptadas y de la caché.	
Observaciones:	

Tabla 2.3 Historia de usuario #3.

Historia de Usuario	
Número: 4	**Nombre historia:** Configurar *in-memory DAO*

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: Edier Campo Pupo	
Descripción: Configurar *in-memory DAO*, debe permitir la configuración del nodo <*bean*> *in-memory DAO*, con todas las propiedades necesarias, de forma fácil y sugerente. Tiene que posibilitar adicionar usuarios, editarlos o eliminarlos. En este caso tiene que ser posible de alguna forma adicionar los roles del proyecto en el que se esté trabajando, para poder crear la lista de usuarios referentes al mismo. Esto debe lograse de manera sencilla y sin necesidad de modificar código o alguna otra característica en la aplicación.	
Observaciones:	

Tabla 2.4 Historia de usuario #4.

		Historia de Usuario
Número: 5	**Nombre historia:** Configurar *JdbcDaoImpl*	
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: Edier Campo Pupo		
Descripción: Configurar *JdbcDaoImpl*, debe permitir la configuración del nodo <*bean*> *JdbcDaoImpl*, con todas las propiedades y configuraciones necesarias, de forma fácil y sugerente. Tiene que permitir reescribir las querys necesarias para posibilitar que sean utilizadas las tablas de la		

base de datos del proyecto que se esté desarrollando, sin necesidad de hacer cambios en las mismas.
Observaciones:

Tabla 2.5 Historia de usuario #5.

Historia de Usuario	
Número: 6	**Nombre historia:** Configurar *anonymousAuthenticationProvider*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 3
Prioridad en negocio: Baja	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *anonymousAuthenticationProvider*, debe permitir la configuración del nodo <*bean*> *anonymousAuthenticationProvider*, con todas las propiedades y configuraciones necesarias, de forma fácil y sugerente.	
Observaciones:	

Tabla 2.6 Historia de usuario #6.

Historia de Usuario	
Número: 7	**Nombre historia:** Configurar *rememberMeAuthenticationProvider*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 3
Prioridad en negocio: Baja	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	

Descripción: Configurar *rememberMeAuthenticationProvider*, debe permitir la configuración del nodo <*bean*> *rememberMeAuthenticationProvider*, con todas las propiedades y configuraciones necesarias, de forma fácil y sugerente.
Observaciones:

Tabla 2.7 Historia de usuario #7.

		Historia de Usuario
Número: 8	**Nombre historia:** Configurar *Access Decision Manager*	
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: Edier Campo Pupo		
Descripción: Configurar *Access Decision Manager*, debe permitir la configuración del nodo <*bean*> principal *accessDecisionManager*, con todas las propiedades y configuraciones necesarias, de forma fácil y sugerente. Debe permitir la selección de cualquiera de las clases proporcionadas por el framework para este nodo, y la configuración de la propiedad *allowIfAllAbstain.*		
Observaciones:		

Tabla 2.8 Historia de usuario #8.

		Historia de Usuario
Número: 9	**Nombre historia:** Configurar *Role Voter*	
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: Edier Campo Pupo	
Descripción: Configurar *Role Voter*, debe permitir la configuración del nodo *<bean> roleVoter*, con todas las propiedades necesarias, de forma fácil y sugerente. Debe permitir el cambio del prefijo *ROLE_* por otro cualquiera que sea necesario en el proyecto que se desarrolla, cumpliendo con el estándar de escritura de los mismos.	
Observaciones:	

Tabla 2.9 Historia de usuario #9.

		Historia de Usuario
Número: 10	**Nombre historia:** Configurar Seguridad de Aplicaciones Web	
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original		
Usuario: Cliente “competente” en el equipo.		**Iteración asignada:** 2
Prioridad en negocio: Alta		**Puntos estimados:**
Riesgo en desarrollo: Media		**Puntos reales:**
Programador responsable: Edier Campo Pupo		
Descripción: Configurar la seguridad de aplicaciones web. Debe permitir la configuración de los mecanismos necesarios para la configuración de las políticas de seguridad en aplicaciones web. Para este caso como prioridad debe controlarse las configuraciones de los mecanismos básicos que tienen que estar presentes en todas las configuraciones de este tipo.		
Observaciones:		

Tabla 2.10 Historia de usuario #10.

Historia de Usuario	
Número: 11	**Nombre historia:** Configurar *FilterToBeanProxy*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *FilterToBeanProxy* en el fichero web.xml, con las propiedades y configuraciones necesarias. Debe ser posible seleccionar el fichero donde se quiere salvar esta configuración.	
Observaciones:	

Tabla 2.11 Historia de usuario #11.

Historia de Usuario	
Número: 12	**Nombre historia:** Configurar *FilterChainProxy*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *FilterChainProxy* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias. Debe permitir adicionar, cambiar y eliminar filtros a la lista de filtros a utilizar. Mantener el orden de los filtros correctamente.	
Observaciones:	

Tabla 2.12 Historia de usuario #12.

Historia de Usuario	
Número: 13	**Nombre historia:** Configurar *httpSessionIntegrationFilter*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *httpSessionIntegrationFilter* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias.	
Observaciones:	

Tabla 2.13 Historia de usuario #13.

Historia de Usuario	
Número: 14	**Nombre historia:** Configurar *Authentication processing mechanisms*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *Authentication processing mechanisms* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias. Para este caso la aplicación tiene que permitir configurar cualquiera de los filtros que Acegi proporciona	

para la autenticación, con su respectivo *entry point,* configurando para cada uno de ellos todas las propiedades necesarias. Deben priorizarse los mecanismos que proporcionan autenticación básica y a través de formularios web personalizados.
Observaciones:

Tabla 2.14 Historia de usuario #14.

Historia de Usuario	
Número: 15	**Nombre historia:** Configurar *ExceptionTranslationFilter*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *ExceptionTranslationFilter* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias. Para este caso debe poder configurarse además la propiedad *AccessDeniedHandlerImpl* y el nodo al cual este referencia, con todas las propiedades y características necesarias.	
Observaciones:	

Tabla 2.15 Historia de usuario #15.

Historia de Usuario	
Número: 16	**Nombre historia:** Configurar *FilterSecurityInterceptor*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2

Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *FilterSecurityInterceptor*, en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias. En este caso debe permitirse editar la lista de URLs que se quiere filtrar.	
Observaciones:	

Tabla 2.16 Historia de usuario #16.

		Historia de Usuario
Número: 17	**Nombre historia:** Configurar *ChannelProcessingFilter*	
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original		
Usuario: Cliente “competente” en el equipo.		**Iteración asignada:** 2
Prioridad en negocio: Alta		**Puntos estimados:**
Riesgo en desarrollo: Media		**Puntos reales:**
Programador responsable: Edier Campo Pupo		
Descripción: Configurar *ChannelProcessingFilter* en el fichero de contexto de la aplicación, con las propiedades y requisitos necesarios. En este caso debe permitirse editar la lista de URLs que necesitan de un canal seguro o no.		
Observaciones:		

Tabla 2.17 Historia de usuario #17.

	Historia de Usuario
Número: 18	**Nombre historia:** Configurar *ChannelDecisionManagerImpl*

Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 2
Prioridad en negocio: Alta	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *ChannelDecisionManagerImpl* en el fichero de contexto de la aplicación, con las propiedades y requisitos necesarios.	
Observaciones:	

Tabla 2.18 Historia de usuario #18.

	Historia de Usuario
Número: 19	**Nombre historia:** Configurar *RunAsManagerImpl*
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 3
Prioridad en negocio: Baja	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *RunAsManagerImpl* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias.	
Observaciones:	

Tabla 2.19 Historia de usuario #19.

	Historia de Usuario
Número: 20	**Nombre historia:** Configurar *Method Invocations*

Modificación (o extensión) de Historia de Usuario (Nro. y Nombre): Historia original	
Usuario: Cliente “competente” en el equipo.	**Iteración asignada:** 3
Prioridad en negocio: Baja	**Puntos estimados:**
Riesgo en desarrollo: Media	**Puntos reales:**
Programador responsable: Edier Campo Pupo	
Descripción: Configurar *Method Invocations* en el fichero de contexto de la aplicación, con las propiedades y configuraciones necesarias.	
Observaciones:	

Tabla 2.20 Historia de usuario #20.

2.5 Diagrama de clases del diseño

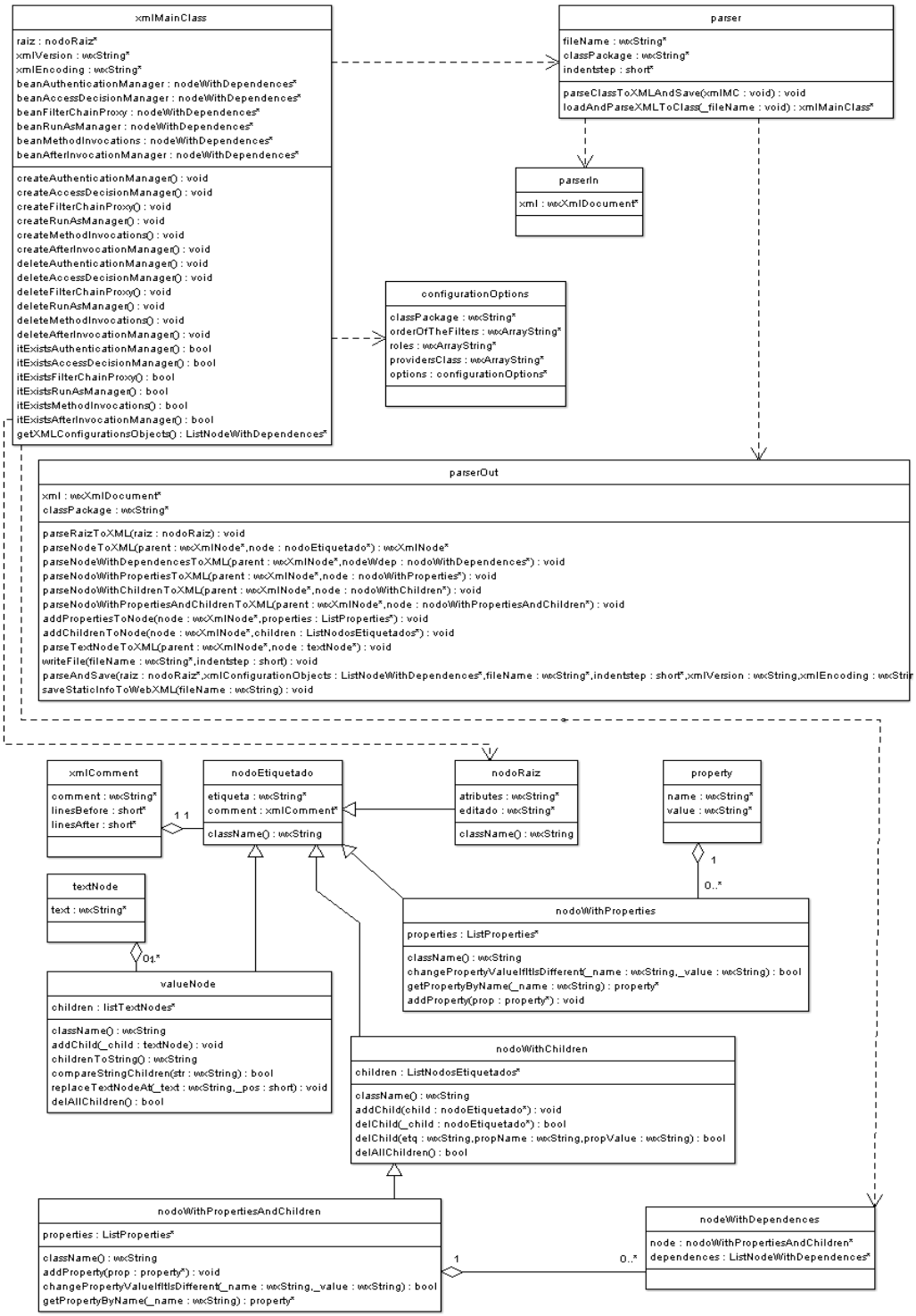


Figura 2.3 Diagrama de clases del diseño.

2.6 Descripción de las clases del diseño

Este epígrafe contiene la descripción de las clases del diseño, para cada una de ellas se muestra su nombre y tipo, una breve explicación de su responsabilidad o función, y el nombre y tipo de cada uno de sus atributos. Se expone además efímeramente el objetivo y forma de funcionamiento de todas las operaciones contenidas por la clase, exceptuando aquellas comúnmente conocidas como “get” y “set” o métodos de acceso a las variables de instancia.

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()`

Descripción	Es el responsable de crear el objeto (nodo bean) AccessDecisonManager, que contiene toda la configuración referente a la decisión de acceso, es el nodo principal de la decisión de acceso y contiene una lista de todos los nodos que colaboran con él.
Nombre	**void** createFilterChainProxy()
Descripción	Es el responsable de crear el objeto (nodo bean) FilterChainProxy, que contiene toda la configuración referente a la seguridad de una aplicación web, básicamente filtros y otros objetos que colaboran con ellos, contiene una lista de todos los nodos que colaboran con él.
Nombre	**void** createRunAsManager()
Descripción	Es el responsable de crear el objeto (nodo bean) RunAsManager, que contiene toda la configuración referente a esta opción de seguridad, es el nodo principal en este sentido y contiene una lista de todos los nodos que colaboran con él.
Nombre	**void** createMethodInvocations()
Descripción	Es el responsable de crear el objeto (nodo bean) MethodInvocations, que contiene toda la configuración referente a la seguridad de aplicaciones de escritorio, contiene una lista de todos los nodos que colaboran con él.
Nombre	**void** createAfterInvocationManager()
Descripción	Es el responsable de crear el objeto (nodo bean) AfterInvocationManager, que contiene toda la configuración referente a esta opción de seguridad, contiene una lista de todos los nodos que colaboran con él.
Nombre	**void** deleteAuthenticationManager()
Descripción	Elimina el objeto AuthenticationManager si este existe y ya no va a ser configurado.
Nombre	**void** deleteAccessDecisionManager()
Descripción	Elimina el objeto AccessDecisionManager si este existe y ya no va a ser configurado.
Nombre	**void** deleteFilterChainProxy()
Descripción	Elimina el objeto FilterChainProxy si este existe y ya no va a ser configurado.

Nombre	**void** deleteRunAsManager()
Descripción	Elimina el objeto RunAsManager si este existe y ya no va a ser configurado.
Nombre	**void** deleteMethodInvocations()
Descripción	Elimina el objeto MethodInvocations si este existe y ya no va a ser configurado.
Nombre	**void** deleteAfterInvocationManager()
Descripción	Elimina el objeto AfterInvocationManager si este existe y ya no va a ser configurado.
Nombre	**bool** itExistsAuthenticationManager()
Descripción	Retorna verdadero si el objeto AuthenticationManager existe, falso en caso contrario.
Nombre	**bool** itExistsAccessDecisionManager()
Descripción	Retorna verdadero si el objeto AccessDecisionManager existe, falso en caso contrario.
Nombre	**bool** itExistsFilterChainProxy()
Descripción	Retorna verdadero si el objeto FilterChainProxy existe, falso en caso contrario.
Nombre	**bool** itExistsRunAsManager()
Descripción	Retorna verdadero si el objeto RunAsManager existe, falso en caso contrario.
Nombre	**bool** itExistsMethodInvocations()
Descripción	Retorna verdadero si el objeto MethodInvocations existe, falso en caso contrario.
Nombre	**bool** itExistsAfterInvocationManager()
Descripción	Retorna verdadero si el objeto AfterInvocationManager existe, falso en caso contrario.
Nombre	ListNodeWithDependences* getXMLConfigurationsObjects()
Descripción	Retorna una lista que contiene todos los objetos que existen (objetos que deben haber sido configurados) de los seis mencionados anteriormente y que representa cada uno la base de la configuración de un aspecto de la seguridad en Acegi. Este método se utilizara básicamente a la hora de salvar la

	configuración seleccionada en el fichero XML.

Tabla 2.21 Descripción de la clase del diseño (`mainClass`).

Nombre	`parser`
Tipo de clase	Controladora
Descripción	
Controla las operaciones de lectura y escritura de los datos, no es la encargada de hacer estas operaciones, sólo las controla, delegando en otras clases la operación en sí. Se encarga de obtener y pasar los atributos necesarios a la clase que realiza la escritura de los datos en el fichero XML para que esta pueda hacer su trabajo, e igual recibe los datos cargados por la clase encargada de hacer la lectura del fichero XML.	
Atributo	**Tipo**
`fileName`	`wxString*`
`classPackage`	`wxString*`
`indentstep`	**`short`***
Para cada responsabilidad	
Nombre	**`void`** `parseClassToXMLAndSave(xmlMainClass* xmlMC)`
Descripción	Recibe el XML representado por objetos, y delega la operación de crear y salvar el fichero XML a la clase parserOut, pasándole todos los parámetros que esta necesite para hacer su trabajo.
Nombre	`xmlMainClass* loadAndParseXMLToClass(wxString _fileName)`
Descripción	Delega a la clase parserIn la tarea de leer un fichero XML determinado, y actualiza la ruta y nombre del fichero que se comenzará a utilizar a partir de la lectura. Recibe como respuesta y a su vez retorna los datos leídos en el formato adecuado.

Tabla 2.22 Descripción de la clase del diseño (`parser`).

Nombre	`parserOut`
Tipo de clase	Acceso a datos

Descripción	
Es la clase encargada de transformar los objetos en nodos XML reales con el formato adecuado, una vez terminado todo este proceso, escribe en el fichero dado el documento XML conformado.	
Atributo	**Tipo**
`xml`	`wxXmlDocument*`
`classPackage`	`wxString*`
Para cada responsabilidad	
Nombre	**`void`** `parseRaizToXML(nodoRaiz *raiz)`
Descripción	Transforma el objeto raíz recibido como parámetro en un nodo XML real, y lo convierte en la raíz del documento XML.
Nombre	`wxXmlNode* parseNodeToXML(wxXmlNode *parent, nodoEtiquetado *node)`
Descripción	Crea un nodo XML real a partir del objeto "node" recibido, este lo agrega al final de la lista de hijos del nodo "parent" y escribe los elementos comunes de cualquier tipo de nodo recibido como la etiqueta y los comentarios. Además inserta una línea en blanco si el "parent" es la raíz para darle mayor legibilidad al documento, al final retorna un apuntador al nuevo nodo.
Nombre	**`void`** `parseNodeWithDependencesToXML(wxXmlNode *parent, nodeWithDependences *nodeWdep)`
Descripción	Se encarga de parsear los objetos que contienen un nodo principal y una lista de nodos de los cuales este depende para realizar sus funciones; él no hace las operaciones directamente, sólo se encarga de delegar responsabilidades a otras funciones.
Nombre	**`void`** `parseNodoWithPropertiesToXML(wxXmlNode *parent, nodoWithProperties *node)`
Descripción	Similar al anterior, pero esta vez para los nodos que sólo contienen una o más propiedades y ningún hijo, igual que el anterior no hace ninguna operación directamente, sino que delega estas a otras funciones.
Nombre	**`void`** `parseNodoWithChildrenToXML(wxXmlNode *parent, nodoWithChildren *node)`
Descripción	Similar a los anteriores, pero para los nodos que sólo contienen hijos y

	ninguna propiedad, igual que los anteriores delega las operaciones en otras funciones.
Nombre	**void** parseNodoWithPropertiesAndChildrenToXML(wxXmlNode *parent, nodoWithPropertiesAndChildren *node)
Descripción	Similar a los anteriores, pero para los nodos que tienen tanto hijos como propiedades, igual que los anteriores delega las operaciones en otras funciones.
Nombre	**void** addPropertiesToNode(wxXmlNode *node, ListProperties *properties)
Descripción	Escribe una o más propiedades en el nodo XML real "node".
Nombre	**void** addChildrenToNode(wxXmlNode *node, ListNodosEtiquetados *children)
Descripción	Identifica cada uno de los hijos recibidos y delega la operación de adicionarlo a la lista de hijos del nodo XML real "node".
Nombre	**void** parseTextNodeToXML(wxXmlNode *parent, textNode *node)
Descripción	Adiciona a la lista de hijos del nodo XML real "parent" el nodo texto recibido.
Nombre	**void** writeFile(wxString *fileName, **short** indentstep)
Descripción	Se encarga de escribir en un dispositivo de almacenamiento físico el XML construido a partir de los objetos.
Nombre	**void** parseAndSave(nodoRaiz *raiz, ListNodeWithDependences *xmlConfigurationObjects, wxString *fileName, **short** *indentstep, wxString xmlVersion="1.0", wxString xmlEncoding="UTF-8")
Descripción	Se encarga de recibir los objetos y otros parámetros que tienen que ver con la estructura del fichero XML, delegando en otros métodos la tarea de construir el XML y escribirlo en un dispositivo de almacenamiento físico.
Nombre	**void** saveStaticInfoToWebXML(wxString fileName)
Descripción	Escribe en el fichero "web.xml" seleccionado la información necesaria para configurar los filtros que se utilicen para asegurar las aplicaciones web.

Tabla 2.23 Descripción de la clase del diseño (parserOut).

Nombre	parserIn
Tipo de clase	Acceso a datos
Descripción	
Es la responsable de leer los datos de un fichero XML y parsearlos, o sea, construir los objetos necesarios para conformar la clase que representa al XML en forma de objeto. Debe reconocer cada una de las configuraciones generales y construir los objetos correspondientes.	
Atributo	**Tipo**
xml	wxXmlDocument*
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.24 Descripción de la clase del diseño (parserIn).

Nombre	nodoEtiquetado
Tipo de clase	Controladora
Descripción	
Es la base para representar un nodo del XML, tiene como atributo la etiqueta correspondiente al nodo.	
Atributo	**Tipo**
etiqueta	wxString*
comment	xmlComment*
Para cada responsabilidad	
Nombre	virtual wxString className()=0
Descripción	Método virtual y abstracto, se encarga de retornar el nombre de la clase, es utilizado para determinar el tipo de hijo con el que se está trabajando.

Tabla 2.25 Descripción de la clase del diseño (nodoEtiquetado).

Nombre	nodoRaiz
Tipo de clase	Controladora

Descripción	
Representa el nodo raíz del documento XML, contiene información importante como direcciones web relacionadas con Acegi, y un atributo especial que determina si un fichero XML que se cargue con la aplicación, fue generado o editado por esta con anterioridad, para que se tenga en cuenta en el proceso de lectura.	
Atributo	**Tipo**
`atributes`	`wxString*`
`editado`	`wxString*`
Para cada responsabilidad	
Nombre	`wxString className()`
Descripción	Implementación del método virtual heredado de la clase "nodoEtiquetado"

Tabla 2.26 Descripción de la clase del diseño (`nodoRaiz`).

Nombre	`textNode`
Tipo de clase	Controladora
Descripción	
Representa un nodo de tipo texto en el XML, este a diferencia de los demás no tiene una etiqueta.	
Atributo	**Tipo**
`text`	`wxString*`
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.27 Descripción de la clase del diseño (`textNode`).

Nombre	`property`
Tipo de clase	Controladora
Descripción	
Representa una propiedad de un nodo, estas tienen un nombre y un valor asociado a dicho nombre.	
Atributo	**Tipo**

name	wxString*
value	wxString*
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.28 Descripción de la clase del diseño (property).

Nombre	configurationOptions
Tipo de clase	Controladora
Descripción	
Almacena un grupo de configuraciones generales que pueden ser editadas por la aplicación, estas son utilizadas en diferentes puntos de la misma para realizar determinadas tareas.	
Atributo	**Tipo**
classPackage	wxString*
orderOfTheFilters	wxArrayString*
roles	wxArrayString*
providersClass	wxArrayString*
options	configurationOptions*
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.29 Descripción de la clase del diseño (configurationOptions).

Nombre	nodeWithDependences
Tipo de clase	Controladora
Descripción	
Es la clase que representa generalmente a los nodos *bean* del XML, prácticamente todos los nodos de este tipo tienen uno o más nodos asociados a él para ayudarlo a cumplimentar sus tareas, en el caso de no depender de nadie su lista de dependencias es NULL. Son la base de cada una de las seis configuraciones generales que debe administrar esta aplicación.	
Atributo	**Tipo**

node	nodoWithPropertiesAndChildren*
dependences	ListNodeWithDependences*
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.30 Descripción de la clase del diseño (nodeWithDependences).

Nombre	nodoWithChildren
Tipo de clase	Controladora
Descripción	
Representa a los tipos de nodos del XML que tienen hijos, pero ninguna propiedad.	
Atributo	**Tipo**
children	ListNodosEtiquetados*
Para cada responsabilidad	
Nombre	wxString className()
Descripción	Implementación del método virtual heredado de la clase nodoEtiquetado.
Nombre	**void** addChild(nodoEtiquetado *child)
Descripción	Adiciona un nuevo hijo al final de la lista de hijos de este objeto.
Nombre	**bool** delChild(nodoEtiquetado *_child)
Descripción	Elimina el nodo recibido como parámetro (en este caso un apuntador al objeto hijo) de la lista de hijos de este objeto retornando verdadero si es posible la operación y falso en otro caso.
Nombre	**bool** delChild(wxString etq, wxString propName, wxString propValue)
Descripción	Sobrecarga de la función anterior, permite eliminar un hijo de la lista de hijos de este objeto buscándolo e identificándolo por su etiqueta, y además por el nombre y valor de una de sus propiedades. Retorna verdadero si es posible la operación, falso en caso contrario.
Nombre	**bool** delAllChildren()

Descripción	Elimina todos los hijos de este objeto, retorna verdadero en caso de ser posible la operación, falso en caso contrario.

Tabla 2.31 Descripción de la clase del diseño (`nodoWithChildren`).

Nombre	`nodoWithProperties`	
Tipo de clase	Controladora	
Descripción		
Representa un nodo del XML que tiene propiedades pero ningún hijo.		
Atributo		**Tipo**
`properties`		`ListProperties*`
Para cada responsabilidad		
Nombre	`wxString className()`	
Descripción	Implementación del método virtual heredado de la clase "nodoEtiquetado".	
Nombre	**`void`** `addProperty(property *prop)`	
Descripción	Adiciona una nueva propiedad al objeto.	
Nombre	**`bool`** `changePropertyValueIfItIsDifferent(wxString _name, wxString _value)`	
Descripción	Busca una propiedad en la lista de propiedades de este objeto comparando por su nombre, retorna verdadero si se cambió el valor por el recibido, falso en cualquier otro caso.	
Nombre	`property* getPropertyByName(wxString _name)`	
Descripción	Busca y retorna una propiedad por su nombre en la lista de propiedades de este objeto, en caso de no encontrarla retorna NULL.	

Tabla 2.32 Descripción de la clase del diseño (`nodoWithProperties`).

Nombre	`nodoWithPropertiesAndChildren`	
Tipo de clase	Controladora	
Descripción		
Representa un nodo del XML que tiene propiedades e hijos.		
Atributo		**Tipo**

properties	ListProperties*
Para cada responsabilidad	
Nombre	wxString className()
Descripción	Implementación del método virtual heredado de la clase "nodoEtiquetado".
Nombre	**void** addProperty(property *prop)
Descripción	Adiciona una nueva propiedad al objeto.
Nombre	**bool** changePropertyValueIfItIsDifferent(wxString _name, wxString _value)
Descripción	Busca una propiedad en la lista de propiedades de este objeto comparando por su nombre, retorna verdadero si se cambió el valor por el recibido, falso en cualquier otro caso.
Nombre	property* getPropertyByName(wxString _name)
Descripción	Busca y retorna una propiedad por su nombre en la lista de propiedades de este objeto, en caso de no encontrarla retorna NULL.

Tabla 2.33 Descripción de la clase del diseño (nodoWithPropertiesAndChildren).

Nombre	valueNode
Tipo de clase	Controladora
Descripción	
Representa a un nodo XML del tipo que no tiene propiedades y su lista de hijos son sólo nodos de tipo texto.	
Atributo	**Tipo**
children	listTextNodes*
Para cada responsabilidad	
Nombre	wxString className()
Descripción	Implementación del método virtual heredado de la clase "nodoEtiquetado".
Nombre	**void** addChild(textNode *_child)
Descripción	Adiciona un nuevo hijo a la lista de hijos de este objeto.
Nombre	wxString childrenToString()
Descripción	Retorna todos los hijos de este objeto en una sola cadena de caracteres.

Nombre	`bool compareStringChildren(wxString str)`
Descripción	Compara si la cadena de caracteres recibida por parámetros es igual a la cadena formada por todos sus hijos, retornando verdadero en caso de cumplirse esto y falso si no.
Nombre	**`bool`** `replaceTextNodeAt(wxString _text, short _pos)`
Descripción	Remplaza el nodo texto en la posición recibida por parámetro, y retorna verdadero si esto es posible, falso en cualquier otro caso.
Nombre	**`bool`** `delAllChildren()`
Descripción	Elimina todos los hijos de este objeto, retorna verdadero si la operación es posible, falso si no.

Tabla 2.34 Descripción de la clase del diseño (`valueNode`).

Nombre	`xmlComment`
Tipo de clase	Controladora
Descripción	
Representa un comentario en el documento XML, este puede contener además del texto del comentario, el número de líneas en blanco que deben insertarse antes y después de él.	
Atributo	**Tipo**
`comment`	`wxString*`
`linesBefore`	**`short`***
`linesAfter`	**`short`***
Para cada responsabilidad	
Nombre	
Descripción	

Tabla 2.35 Descripción de la clase del diseño (`xmlComment`).

2.7 Tareas de ingeniería

En el actual epígrafe se exponen las “Tareas de ingeniería”, otro de los artefactos propuestos por la metodología XP (anexo III). Estas contribuyen a organizar el proceso de desarrollo mediante la planificación del tiempo que se estima debe tardar la implementación de una o varias historias de usuario.

Una tarea de ingeniería debe incluir el nombre del responsable de su ejecución, que puede ser cualquier persona que ocupe un rol que coincida con el objetivo de la tarea y pueda cumplirla en el plazo estimado.

Tarea de ingeniería	
Número tarea: 1	**Número historia:** 1
Nombre tarea: Diseño interfaz primaria. Ventana Acegi-VisualConfig *(Authentication Manager).	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 17/05/2008	**Fecha fin:** 18/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se desarrollará una ventana que contendrá la presentación inicial de la aplicación y el punto de partida para el inicio de la configuración de la autenticación, teniendo en cuenta que debe ser posible ir agregando progresivamente, el resto de los componentes visuales que permitirán el inicio de las demás configuraciones, sin que sea necesario un cambio en lo ya diseñado.	

Tabla 2.36 Tarea de ingeniería #1.

Tarea de ingeniería	
Número tarea: 2	**Número historia:** 2
Nombre tarea: Diseño interfaz secundaria. Ventana Authentication Manager.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 18/05/2008	**Fecha fin:** 19/05/2008
Programador responsable: Edier Campo Pupo	
Descripción:	

Se desarrollará una ventana que contendrá una lista de todos los providers que pueden ser configurados, permitiendo la selección de uno, varios o todos. Debe ofrecer la posibilidad de configurar cada provider seleccionado.

Tabla 2.37 Tarea de ingeniería #2.

Tarea de ingeniería	
Número tarea: 3	**Número historia:** 2, 4, 5
Nombre tarea: Diseño interfaz secundaria. Ventanas de configuración de los providers.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 19/05/2008	**Fecha fin:** 20/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se desarrollarán una serie de ventanas que permitirán configurar cada uno de los providers seleccionados, y además todas las ventanas que sean necesarias para la configuración de aspectos o propiedades propias de cada provider.	

Tabla 2.38 Tarea de ingeniería #3.

Tarea de ingeniería	
Número tarea: 4	**Número historia:** 8, 9
Nombre tarea: Diseño interfaz primaria. Ventana Acegi-VisualConfig *(Access Decision Manager).	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 20/05/2008	**Fecha fin:** 21/05/2008

Programador responsable: Edier Campo Pupo
Descripción: Se agregará a la ventana principal de la aplicación el punto inicial para la configuración de Access Decision Manager, teniendo en cuenta no modificar el diseño ya existente, y además se diseñarán las ventanas necesarias para configurar Role Voter.

Tabla 2.39 Tarea de ingeniería #4.

	Tarea de ingeniería
Número tarea: 5	**Número historia:** 10
Nombre tarea: Diseño interfaz primaria. Ventana Acegi-VisualConfig *(Web Applications).	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 21/05/2008	**Fecha fin:** 22/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se agregará a la ventana principal de la aplicación el punto inicial para la configuración de Web Applications, teniendo en cuenta no modificar el diseño ya existente.	

Tabla 2.40 Tarea de ingeniería #5.

	Tarea de ingeniería
Número tarea: 6	**Número historia:** 12, 13, 14, 15, 16
Nombre tarea: Diseño interfaz secundaria. Ventana Filters.	
Tipo de tarea: Desarrollo	**Puntos estimados:**

Fecha inicio: 22/05/2008	**Fecha fin:** 23/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se agregará una nueva ventana a la aplicación que permita la configuración de los filtros, principalmente aquellos que deben estar por obligación en todas las aplicaciones de este tipo, y además posibilitando configurar aquellos que se consideren sean necesarios.	

Tabla 2.41 Tarea de ingeniería #6.

	Tarea de ingeniería
Número tarea: 7	**Número historia:** 1, 2, 3, 4, 5
Nombre tarea: Implementación**,** Authentication Manager.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 23/05/2008	**Fecha fin:** 27/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementan todas las clases necesarias para permitir el funcionamiento de esta parte de la aplicación.	

Tabla 2.42 Tarea de ingeniería #7.

	Tarea de ingeniería
Número tarea: 8	**Número historia:** 8, 9
Nombre tarea: Implementación**,** Access Decision Manager.	

Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 27/05/2008	**Fecha fin:** 29/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementan todas las clases necesarias para permitir el funcionamiento de esta parte de la aplicación.	

Tabla 2.43 Tarea de ingeniería #8.

	Tarea de ingeniería
Número tarea: 9	**Número historia:** 11
Nombre tarea: Implementación, salvar en web.xml.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 29/05/2008	**Fecha fin:** 29/05/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementa la posibilidad de salvar la configuración necesaria en el fichero web.xml que se seleccione para esta historia de usuario.	

Tabla 2.44 Tarea de ingeniería #9.

	Tarea de ingeniería
Número tarea: 10	**Número historia:** 10, 11, 12, 13, 14, 15, 16
Nombre tarea: Implementación**,** Web Applications.	

Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 30/05/2008	**Fecha fin:** 3/06/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementan todas las clases necesarias para permitir el funcionamiento de esta parte de la aplicación.	

Tabla 2.45 Tarea de ingeniería #10.

	Tarea de ingeniería
Número tarea: 11	**Número historia:** 17, 18
Nombre tarea: Implementación**,** Web Applications más filtros.	
Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 3/06/2008	**Fecha fin:** 5/06/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementan todas las clases necesarias para ampliar el funcionamiento de esta parte de la aplicación.	

Tabla 2.46Tarea de ingeniería #11.

	Tarea de ingeniería
Número tarea: 12	**Número historia:** 19, 20, 6, 7
Nombre tarea: Implementación**,** otras funcionalidades.	

Tipo de tarea: Desarrollo	**Puntos estimados:**
Fecha inicio: 5/06/2008	**Fecha fin:** 10/06/2008
Programador responsable: Edier Campo Pupo	
Descripción: Se implementan el resto de las funcionalidades que falten.	

Tabla 2.47 Tarea de ingeniería #12

2.8 Interfaz gráfica del sistema

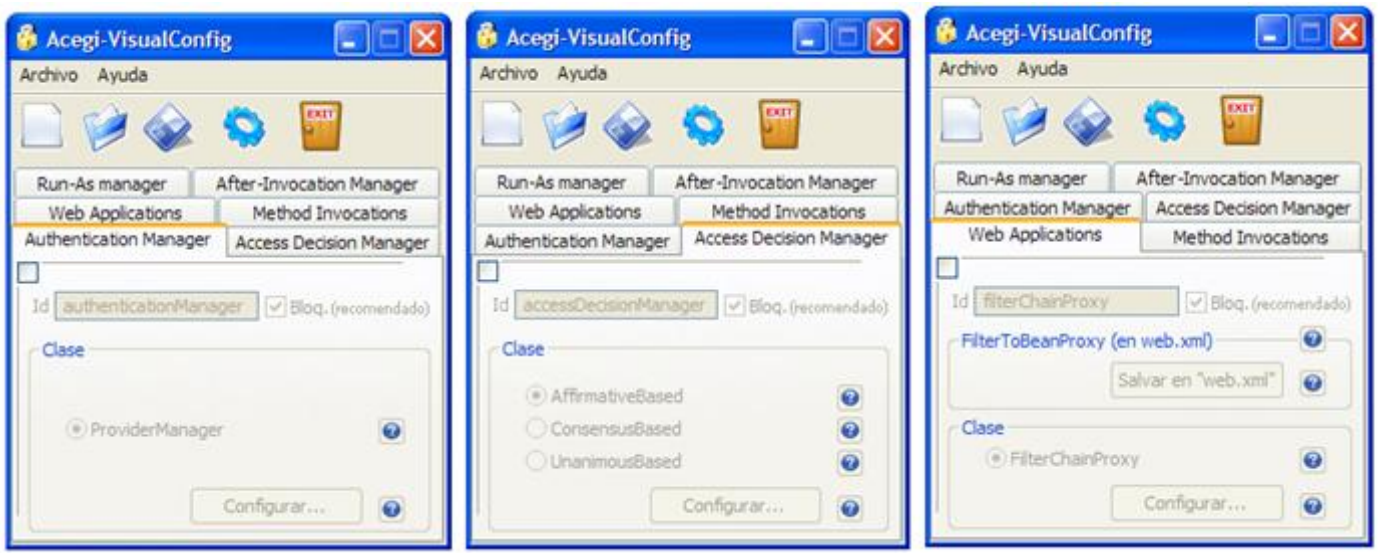


Figura 2.4 Ventana principal (Authentication Manager, Access Decision Manager, Web Applications).

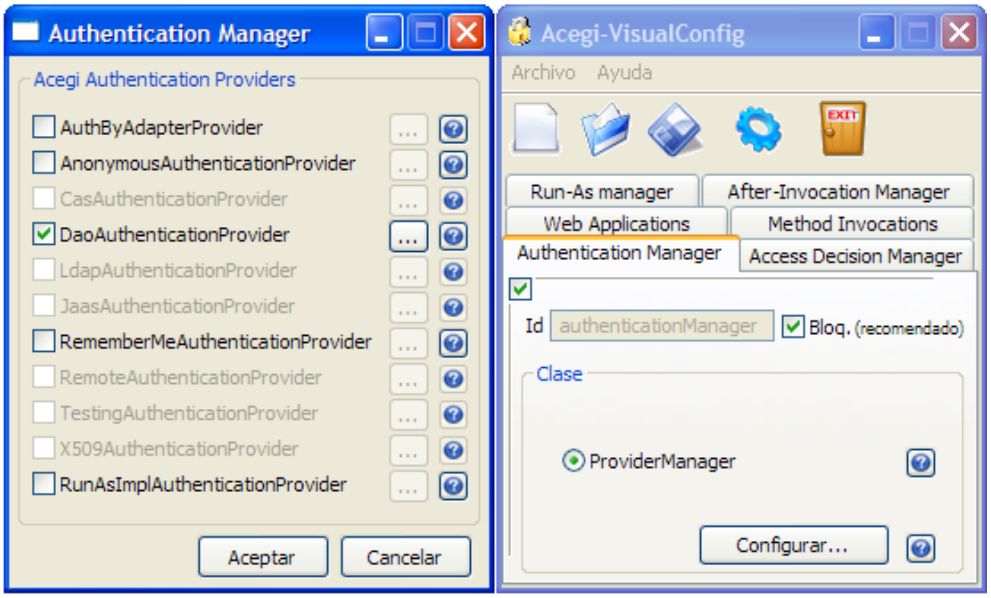


Figura 2.5 Ventana Authentication Manager (seleccionar providers).

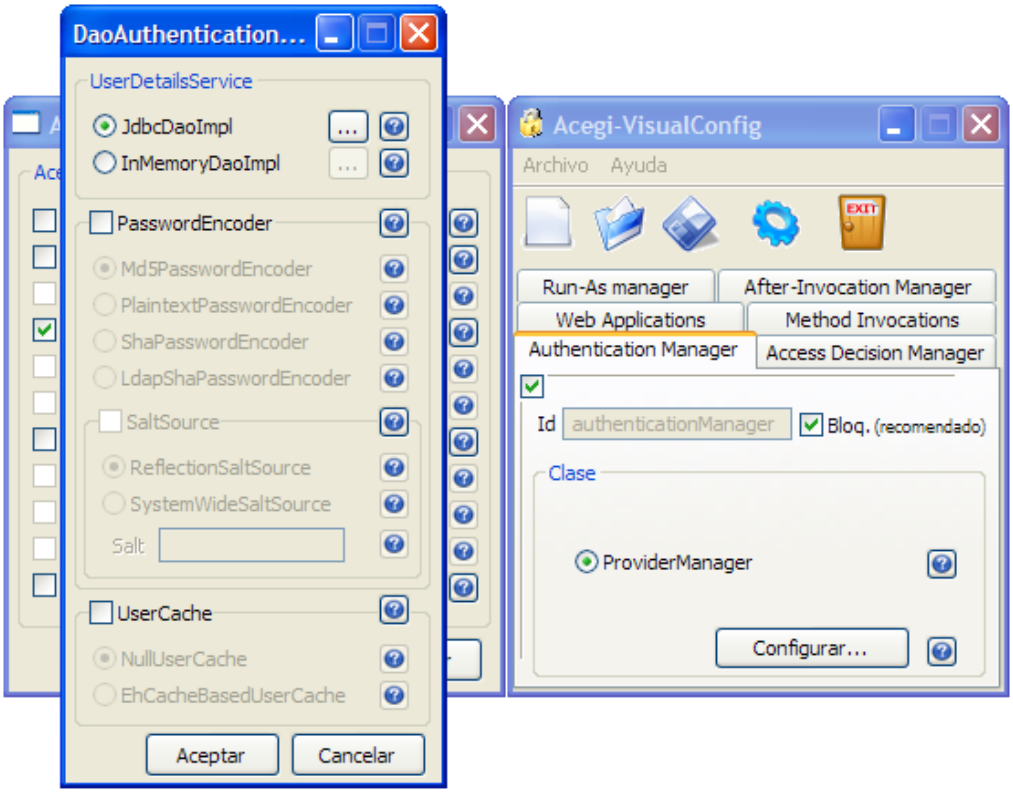


Figura 2.6 Ventana DaoAuthenticationProvider (configurar).

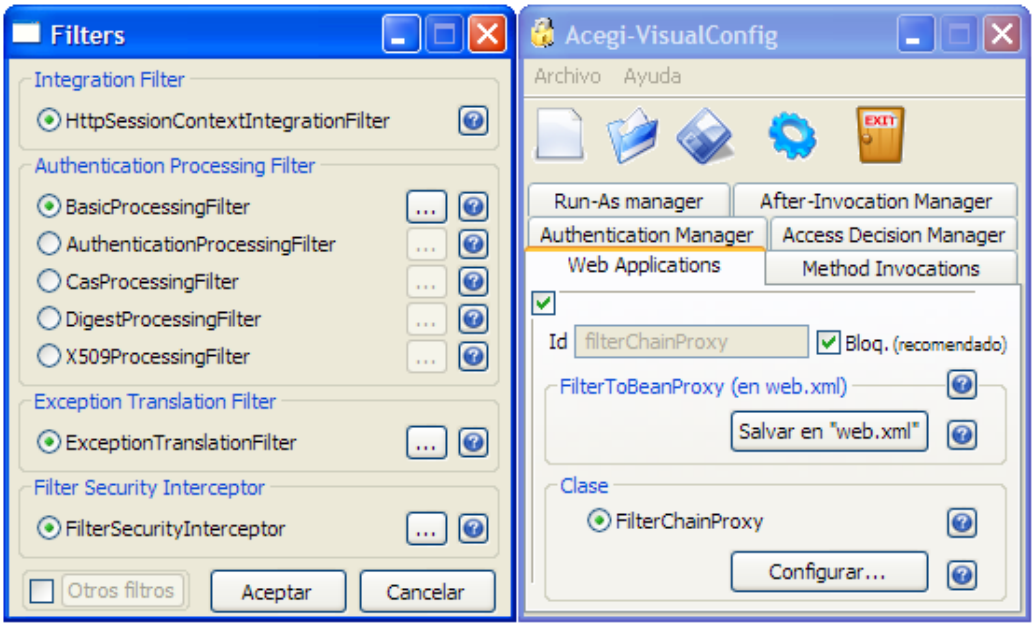


Figura 2.7 Ventana Filters (configurar).

Conclusiones

- La utilización de una arquitectura sencilla en tres capas, permitió separar conceptualmente los procesos a modelar en tres grupos bien delimitados, partiendo de las características y responsabilidades de cada uno de ellos.
- La utilización del patrón de diseño Singleton, aportó claridad y confianza al producto, evitando pasar constantemente información por parámetros, y garantizando que los datos de configuración necesarios, accedidos en cualquier punto de la aplicación, son los requeridos y están correctamente actualizados.
- El empleo de eXtreme Programming como guía de desarrollo, permitió un rápido y flexible entendimiento con el cliente, debido a que a diferencia de otras metodologías, el lenguaje utilizado en todo el proceso es formal y sin tecnicismos. La utilización de esta metodología aportó rapidez en el proceso de implementación.
- El modelado del diagrama de clases del diseño contribuyó a un mejor entendimiento del sistema, y por tanto a su mejoramiento, proporcionando un punto de vista global de las características del sistema.

- La descripción de cada una de las clases del diseño favoreció la comprensión de las responsabilidades asignadas de forma mucho más clara, permitiendo la reorganización o reestructuración del diseño cuando fuese necesario.

CAPÍTULO 3: ANÁLISIS DE LOS RESULTADOS

Entre los objetivos del presente capítulo se encuentra la validación de la propuesta de diseño para la aplicación descrita en el apartado anterior, centrándose fundamentalmente en la medición de su calidad. Para esto son aplicadas un grupo de métricas técnicas especialmente enfocadas en la evaluación de sistemas orientados a objetos. 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 (anexo I), comprobando así que las necesidades del cliente fueron satisfechas.

3.1 Métricas aplicadas

En este epígrafe, basado en la importancia del uso de métricas como medidas de legibilidad, reutilización, limpieza y buenas prácticas de programación, se aplican algunas de ellas con el propósito de lograr una validación del diseño e implementación utilizados para dar solución al problema planteado. Partiendo de que el diseño propuesto es un diseño orientado a objetos, se resolvió aplicar métricas concebidas para evaluar los más importantes aspectos de este paradigma, y están especialmente enfocadas a la medición y comprobación de los atributos de calidad propuestos en el capítulo dos.

Las métricas técnicas proporcionan una base desde la cual el análisis, el diseño y o la verificación pueden conducirse de manera más objetiva, y ser evaluados más cuantitativamente. (19)

Las métricas orientadas a objetos se han introducido como ayuda al ingeniero del software en el uso del análisis cuantitativo, para evaluar la calidad en el diseño antes de que un sistema se construya. El enfoque de las métricas orientadas a objetos está en la clase, piedra fundamental en la arquitectura orientada a objetos. (19)

Los objetivos de las métricas orientadas a objeto están centrados principalmente en tres puntos:

- Entender mejor la calidad del producto.
- Evaluar la efectividad del proceso.
- Mejorar la calidad del trabajo llevado a cabo al nivel del proyecto. (19)

Las métricas para cualquier producto de ingeniería son reguladas por las características únicas del producto. El software orientado a objetos es fundamentalmente diferente del software desarrollado con el uso de métodos convencionales. Por esta razón, las métricas para sistemas orientados a objetos deben ser afinadas a las características que distinguen al software orientado a objeto del software convencional. (19)

3.1.1 Métricas CK

Se conocen como métricas CK el conjunto de seis métricas de productos específica para código orientado a objetos propuestas por los autores Chidamber y Kemerer en su publicación "A Metrics Suite for Object Oriented Design" de 1994.

La serie de métricas CK 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 OO, 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. (20)

Profundidad del árbol de herencia (DIT)

La métrica DIT de una clase C es su profundidad en el árbol de herencia (20), 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. (19)

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.

Clase	LOCM
valueNode	2
nodoRaiz	2
nodoWithProperties	2
nodoWithChildren	2
nodoWithPropertiesAndChildren	3

Tabla 3.1 Resultados de la métrica DIT aplicada a las clases hijas (que heredan) del sistema.

Los resultados de aplicar esta métrica a las clases hijas (que heredan) del sistema propuesto (anexo VII), proporcionan una medida de la complejidad del diseño. Como se puede apreciar, el mayor valor DIT obtenido es 3 (tabla 3.1), 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 (20). 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. A medida que el NOC crece, la cantidad de pruebas (requeridas para ejercitar cada descendiente en su contexto operativo) se incrementará también. (19)

NOC ofrece una idea sobre la calidad de la herencia entre clases en el diseño evaluado, permitiendo detectar posibles errores relacionados con esta propiedad. La métrica NOC es aplicable solamente a clases que sirven de base a otras en el proceso de herencia.

Clase	NOC
nodoEtiquetado	4
nodoWithChildren	1

Tabla 3.2 Resultados de la métrica NOC aplicada a las clases base (herencia) del sistema.

Como resultado de aplicar la métrica NOC a la herramienta (tabla 3.2) se tiene que el su máximo nivel es de 4 y corresponde a la clase nodoEtiquetado (anexo VII), valor que demuestra la existencia de un buen diseño de clases y una jerarquía bien estructurada.

Falta de cohesión en los métodos (LOCM)

Cada método dentro de una clase C, accede a uno o varios atributos (también llamados variables de instancia), 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. (19)

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 (anexo VIII).

LOCM no es aplicable a clases con pocos métodos y atributos, o bien los resultados de su aplicación a estas no aportarían ningún indicio importante que permita tomar decisiones sobre el diseño evaluado.

Clase	LOCM
xmlMainClass	3
parserOut	4

Tabla 3.3 Resultados de la aplicación de la métrica LOCM a dos de las principales clases del sistema.

Los resultados obtenidos (tabla 3.3) 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.

3.1.2 Métricas LK

Se conocen como métricas LK el conjunto de métricas propuestas por los autores Lorenz y Kidd en su libro Object-Oriented Software Metrics, publicado el 29 de junio de 1994.

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. (21) (19)

Parámetros de calidad	Valores Grandes de TC
Reutilización	Reduce la reutilización de la clase
Implementación	Complica la implementación
Complejidad de las pruebas	Hace compleja las pruebas del sistema
Responsabilidad	La clase debe tener bastante responsabilidad

Tabla 3.4 Parámetros de calidad para valores grandes de TC. (21)

TC	Umbral
Pequeño	<= 20
Medio	>20 y <=30
Grande	>30

Tabla 3.5 Umbrales aplicados para TC. (21)

No.	Clase	Atributos	Operaciones	Tamaño		
1	xmlMainClass	9	19	Medio		
2	configurationOptions	5	0	Pequeño		
3	parser	3	2	Pequeño		
4	parserOut	2	12	Pequeño		
5	nodoEtiquetado	2	1	Pequeño		
6	xmlCommnet	3	0	Pequeño		
7	textNode	1	0	Pequeño		
8	valueNode	1	6	Pequeño		
9	nodoRaiz	2	1	Pequeño		
10	property	2	0	Pequeño		
11	nodoWithProperties	1	4	Pequeño		
12	nodoWithChildren	1	5	Pequeño		
13	nodoWithPropertiesAndChildren	1	4	Pequeño		
14	nodeWithDependences	2	0	Pequeño		
Total / **Media**		35 / 2.5	54 / 3.85	**P** / 13	**M** / 1	**G** / 0

Tabla 3.6 Clases a las que se les aplicó la métrica TC.

Se le aplicó la métrica TC a un número de 14 clases (tabla 3.6) 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 grande.

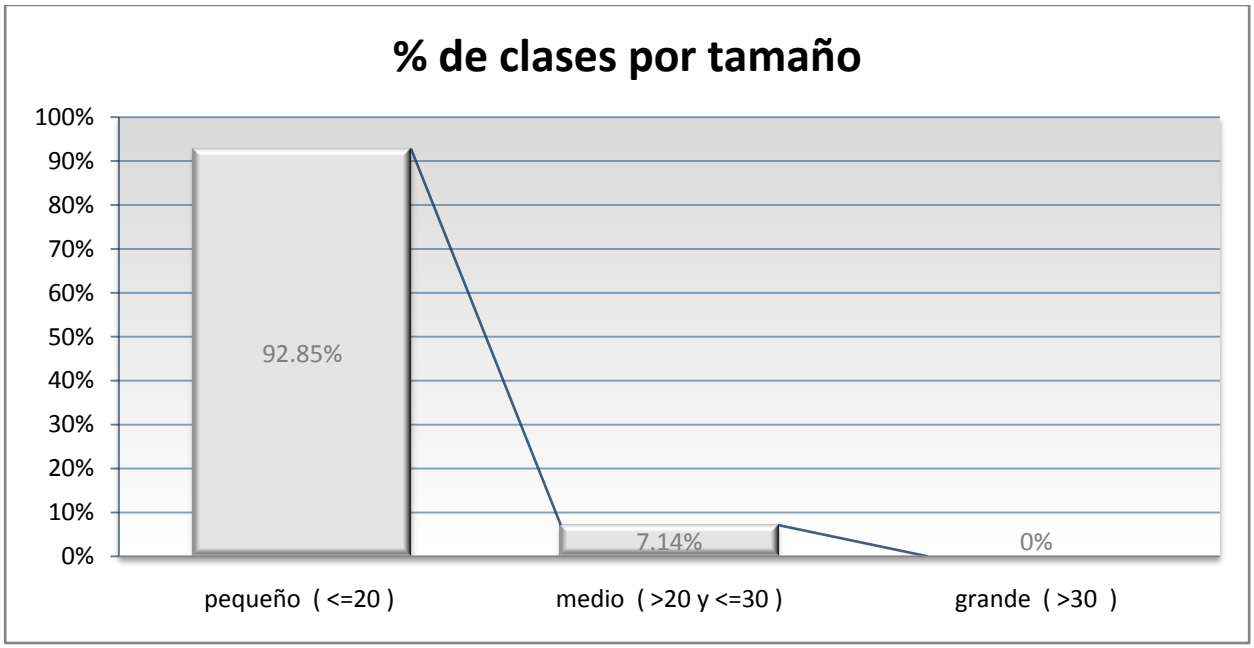


Gráfico 3.1 Porciento de clases por tamaño.

A partir de los resultados anteriores puede apreciarse que 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)

Se define como NOI 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. (19)

Como resultado de aplicar esta métrica al sistema (tabla 3.7), 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.

No.	Clase	Operaciones heredadas	Operaciones redefinidas (NOI)
1	valueNode	1	1
2	nodoRaiz	1	1
3	nodoWithProperties	1	1
4	nodoWithChildren	1	1
5	nodoWithPropertiesAndChildren	5	1

Tabla 3.7 Subclases a las que se le aplicó la métrica NOI.

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.

$IE = (NOI \times nivel)/M_{total}$

En donde “nivel” es el nivel de la jerarquía de clases en que reside la clase, y M_{total} 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.

No.	Clase	NOI	Nivel	M_{total}	IE [$IE = (NOI \times Nivel)/M_{total}$]
1	valueNode	1	1	6	0.16
2	nodoRaiz	1	1	1	1
3	nodoWithProperties	1	1	4	0.25
4	nodoWithChildren	1	1	5	0.2
5	nodoWithPropertiesAndChildren	1	2	8	0.25

Tabla 3.8 Subclases a las que se le aplicó la métrica IE.

Al analizar los resultados arrojados por la aplicación de la métrica IE a las subclases del sistema (tabla 3.8), se infiere que la jerarquía de clases de forma general se ajusta a la abstracción de las superclases.

3.2 Pruebas de aceptación

En este punto la metodología propone un caso de prueba de aceptación por cada una de las historias de usuario (anexo II). 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.

Caso de prueba de aceptación	
Código: 01	**Historia de Usuario:** #1 Configurar *Authentication Managers*
Nombre: *Authentication Managers*	
Descripción: Se configura visualmente *AuthenticationManager*, se cambia la selección de los *providers*, y el id del *bean*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se comprueba que la lista de *providers* y el id del *bean* coincidan con los seleccionados.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.9 Prueba de aceptación #1.

Caso de prueba de aceptación	
Código: 02	**Historia de Usuario:** #2 Configurar *providers*
Nombre: *Providers*	
Descripción: Se configuran los *providers* necesarios, seleccionando y configurando cada uno de los que se desee.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica la lista de	

providers y sus configuraciones.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.10 Prueba de aceptación #2.

Caso de prueba de aceptación	
Código: 03	**Historia de Usuario:** #3 Configurar *daoAuthenticationProvider*
Nombre: *daoAuthenticationProvider*	
Descripción: Se configura daoAuthenticationProvider, se configuran y se cambian las propiedades de este.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se debe encontrar correctamente escrita toda la configuración, verificándose que exista correspondencia entre las propiedades y valores configurados para estas y los escritos.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.11 Prueba de aceptación #3.

Caso de prueba de aceptación	
Código: 04	**Historia de Usuario:** #4 Configurar *in-memory DAO*
Nombre: *In-memory DAO*	
Descripción: Se configura *InMemoryDaoImpl*, se adicionan o eliminan usuarios y propiedades de estos.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que exista correspondencia entre la lista de usuarios seleccionados y sus propiedades y lo escrito.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.12 Prueba de aceptación #4.

Caso de prueba de aceptación	
Código: 05	**Historia de Usuario:** #5 Configurar *JdbcDaoImpl*
Nombre: *JdbcDaoImpl*	
Descripción: Se configura *JdbcDaoImpl*, se reescriben las querys de SQL.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se comprueba que las querys SQL estén correctamente escritas y coincidan con lo seleccionado.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.13 Prueba de aceptación #5.

Caso de prueba de aceptación	
Código: 06	**Historia de Usuario:** #8 Configurar *Access Decision Manager*
Nombre: *Access Decision Manager*	
Descripción: Se configura *AccessDecisionManager*, se cambia la selección de la clase a utilizar y el id del *bean*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que el nombre, ruta de la clase e id del *bean* coincidan con los seleccionados.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.14 Prueba de aceptación #6.

Caso de prueba de aceptación	
Código: 07	**Historia de Usuario:** #9 Configurar *Role Voter*
Nombre: *Role Voter*	

Descripción: Se configura *roleVoter*, se cambia el prefijo, y se configura la propiedad *allowIfAllAbstain*.
Condiciones de Ejecución: Normales.
Entrada / Pasos de ejecución: Selección visual de la configuración.
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que el prefijo coincida con el seleccionado y que la propiedad *allowIfAllAbstain* este configurada según lo seleccionado.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.15 Prueba de aceptación #7.

Caso de prueba de aceptación	
Código: 08	**Historia de Usuario:** #10 Configurar Seguridad de Aplicaciones Web
Nombre: Aplicaciones Web	
Descripción: Se configuran los valores y propiedades que se necesiten, se cambia el id del *bean*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que el id del *bean* coincida con el seleccionado.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.16 Prueba de aceptación #8.

Caso de prueba de aceptación	
Código: 09	**Historia de Usuario:** #11 Configurar *FilterToBeanProxy*
Nombre: *FilterToBeanProxy*	
Descripción: Se configura *FilterToBeanProxy*, se selecciona el fichero *web.xml* donde se almacenara esta configuración.	

Condiciones de Ejecución: Normales.
Entrada / Pasos de ejecución: Selección visual de la configuración.
Resultado Esperado: Al salvar en el fichero *web.xml* seleccionado se encuentre correctamente escrita toda la configuración, se verifica que no sea repetida esta configuración si ya existía en el fichero.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.17 Prueba de aceptación #9.

Caso de prueba de aceptación	
Código: 10	**Historia de Usuario:** #12 Configurar *FilterChainProxy*
Nombre: *FilterChainProxy*	
Descripción: Se configura la lista de filtros y el orden en que serán utilizados.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que los filtros escritos coinciden en nombre y orden con los seleccionados.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.18 Prueba de aceptación #10.

Caso de prueba de aceptación	
Código: 11	**Historia de Usuario:** #13 Configurar *httpSessionIntegrationFilter*
Nombre: *httpSessionIntegrationFilter*	
Descripción: Se configura *httpSessionIntegrationFilter*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración.	

Evaluación de la Prueba: Satisfactoria.

Tabla 3.19 Prueba de aceptación #11.

Caso de prueba de aceptación	
Código: 12	**Historia de Usuario:** #14 Configurar *Authentication processing mechanisms*
Nombre: *Authentication processing mechanisms*	
Descripción: Se selecciona el tipo de mecanismo a utilizar, y se configura este.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que se utilice la pareja (*filter-EntryPonit*) correcta, y su coincidencia con el mecanismo seleccionado.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.20 Prueba de aceptación #12.

Caso de prueba de aceptación	
Código: 13	**Historia de Usuario:** #15 Configurar *ExceptionTranslationFilter*
Nombre: *ExceptionTranslationFilter*	
Descripción: Se configura *ExceptionTranslationFilter*, y la propiedad *accessDeniedHandler*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica que la propiedad *accessDeniedHandler* coincida con la configuración seleccionada.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.21 Prueba de aceptación #13.

Caso de prueba de aceptación	
Código: 14	**Historia de Usuario:** #16 Configurar *FilterSecurityInterceptor*
Nombre: *FilterSecurityInterceptor*	
Descripción: Se configura *FilterSecurityInterceptor*, editando y modificando la lista de URLs que serán interceptadas.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, se verifica la correspondencia entre las URLs escritas y las seleccionadas.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.22 Prueba de aceptación #14.

Caso de prueba de aceptación	
Código: 15	**Historia de Usuario:** #17 Configurar *ChannelProcessingFilter*
Nombre: *ChannelProcessingFilter*	
Descripción: Se configura *ChannelProcessingFilter*, configurando la lista de URLs que necesitan canal seguro o inseguro.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, es verificada la coincidencia de las URLs y el canal que necesitan con lo escrito en el fichero.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.23 Prueba de aceptación #15.

Caso de prueba de aceptación	
Código: 16	**Historia de Usuario:** #18 Configurar *ChannelDecisionManagerImpl*

Nombre: *ChannelDecisionManagerImpl*
Descripción: Se configura *ChannelDecisionManagerImpl*.
Condiciones de Ejecución: Normales.
Entrada / Pasos de ejecución: Selección visual de la configuración.
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, es revisada la lista de clases utilizadas para la lista de canales.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.24 Prueba de aceptación #16.

	Caso de prueba de aceptación
Código: 17	**Historia de Usuario:** #6 Configurar *anonymousAuthenticationProvider*
Nombre: *anonymousAuthenticationProvider*	
Descripción: Se configura *anonymousAuthenticationProvider*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, y esta coincida con lo seleccionado.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.25 Prueba de aceptación #17.

	Caso de prueba de aceptación
Código: 18	**Historia de Usuario:** #7 Configurar *rememberMeAuthenticationProvider*
Nombre: *rememberMeAuthenticationProvider*	
Descripción: Se configura *rememberMeAuthenticationProvider*.	

Condiciones de Ejecución: Normales.
Entrada / Pasos de ejecución: Selección visual de la configuración.
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, con las propiedades y valores seleccionados, las rutas de clases, y la estructura requerida.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.26 Prueba de aceptación #18.

Caso de prueba de aceptación	
Código: 19	**Historia de Usuario:** #19 Configurar *RunAsManagerImpl*
Nombre: *RunAsManagerImpl*	
Descripción: Se configura *RunAsManagerImpl*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado: Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, con las propiedades y valores seleccionados, las rutas de clases, y la estructura requerida.	
Evaluación de la Prueba: Satisfactoria.	

Tabla 3.27 Prueba de aceptación #19.

Caso de prueba de aceptación	
Código: 20	**Historia de Usuario:** #20 Configurar *RunAsImplAuthenticationProvider*
Nombre: *RunAsImplAuthenticationProvider*	
Descripción: Se configura *RunAsImplAuthenticationProvider*.	
Condiciones de Ejecución: Normales.	
Entrada / Pasos de ejecución: Selección visual de la configuración.	
Resultado Esperado:	

Al salvar el fichero XML se encuentre correctamente escrita toda la configuración, con las propiedades y valores seleccionados, las rutas de clases, y la estructura requerida.
Evaluación de la Prueba: Satisfactoria.

Tabla 3.28 Prueba de aceptación #20.

Conclusiones

- Partiendo de los resultados obtenidos al aplicar un grupo de métricas se pudo evaluar el diseño de la solución propuesta en el capítulo dos.
- Se comprobó que el diseño propuesto presenta un bajo nivel de complejidad estructural, facilitando la preparación y aplicación de pruebas al sistema.
- La aplicación de métricas como "Tamaño de clases (TC)" y "Número de operaciones redefinidas (NOI)" evidencian que las clases en su mayoría son reutilizables, y que su implementación no es compleja.
- La profundidad de los niveles de herencia está acorde con los umbrales definidos por algunos autores, favoreciendo el bajo acoplamiento y la poca complejidad del sistema.
- El sistema es robusto, sencillo y fácil de mantener o modificar.
- La aplicación de los "Caso de prueba de aceptación" permitió determinar que la aplicación implementada cumple con las expectativas del cliente.

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.
- Los resultados del estudio de la utilización del framework Acegi en los proyectos productivos de la facultad 3 y de la UCI, sirvieron para enriquecer y mejorar el desarrollo de este trabajo, permitiendo además adicionarle características necesarias para generalizar su fácil empleo.
- El resultado de 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.

RECOMENDACIONES

- Continuar y mejorar el desarrollo de la aplicación, incluyéndole la posibilidad de manejar o editar otras funcionalidades aportadas por el framework Acegi, y que no fueron incluidas en esta primera versión.
- Proporcionarle a la aplicación la característica de determinar si la configuración general del XML no está completa, está 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.
- 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. [En línea] Microsystems, S. Acerca de la tecnología Java. cited from www.java.com/es/about.

2. **Allamaraju, S.** *Programación Java Server con J2EE Edición 1.3.* s.l. : A. Multimedia, 2002.

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

4. **Man, K.D.** *Java Server Faces in Action.* s.l. : M.P. Co., 2005.

5. **Walls, C.** *Spring in Action.* s.l. : M.P. Co., 2005.

6. **BAUER, C.** *Hibernate in Action.* s.l. : M.P. Co, 2005.

7. [En línea] oness.sourceforge.net/JavaHispano%20Acegi.pdf.

8. **Johnson, R.** Introduction to the Spring Framework. [En línea] Mayo de 2005. http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework.

9. [En línea] sentidoweb.com/2006/12/26/spring-framework-de-java.php.

10. **Harrop, R. Machacek, J.** *Pro Spring.* s.l. : Apress, 2005.

11. [En línea] acegisecurity.sourceforge.net.

12. [En línea] www.springframework.org.

13. [En línea] www.javalobby.org/articles/acegisecurity/part1.jsp?source=archives.

14. [En línea] tecnoblog.entel.es/wp-content/uploads/2007/05/introduccion-a-acegi.pdf.

15. **Yale, Universidadde.** CAS (Central Authentication Service). [En línea] http://www.yale.edu/tp/auth/.

16. **Walls, C. Breydenbach, Ryan.** *Spring in Action second edition.* s.l. : M.P. Co., 2008.

17. A propósito de programación extrema XP (eXtreme Programming). [En línea] http://www.monografias.com/trabajos51/programacion-extrema/programacion-extrema.shtml.

18. Metodologias RUP y XP - [PROCESOS DE DESAROLLO]. [En línea] Sábado 5 de Mayo de 2007. http://jackopc.blogspot.com/.

19. **Pressman, Roger S.** *Ingenería de Software. Un enfoque práctico. Vol. I.* 1998.

20. **Arregui, Juan José Olmedilla.** *Revisión Sistemática de Métricas de Diseño Orientado a Objetos.* s.l. : Universidad Politécnica de Madrid, Facultad.de Informática, Septiembre de 2005.

21. **Lorenz, M y Kidd, J.** *Object-Oriented Software Metric.* 1994.

22. [En línea] ame.endesa.es/confluence/display/AMEBASE/ACEGI.

23. [En línea] oness.sourceforge.net/JavaHispano%20Acegi%20presentacion.pdf.

24. [En línea] www.sg.com.mx/content/view/406.

25. [En línea] javahispano.net/frs/shownotes.php?release_id=86.

26. [En línea] www.javaconganas.com/space/path/frameworks/spring/ioc.

27. [En línea] www.javahispano.org/contenidos.item.action?id=3167000&menuId=NEWS.

28. [En línea] www.tiobe.com/index.php/content/paperinfo/tpci/index.html.

29. [En línea] www.palermo.edu/ingenieria/downloads/introduccion_spring_framework_v1.0.pdf.

30. [En línea] javalangnullpointer.wordpress.com/2007/05/21/tecnicas-de-programacion-inversion-de-control.

31. [En línea] www.programacion.net/java/articulo/jap_injection.

32. **Von Bertalanffy, Ludwig.** *Teoría General de los Sistemas.* s.l. : Madrid: Alianza Editorial., 1981.

33. **Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu.** *Professional Java Development with the Spring Framework.* Indianapolis : Wiley Publishing, Inc., 2005.

34. **González, C.S.** ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras. [En línea] 2004. http://oness.sourceforge.net/proyecto/html/index.html.

35. **Johnson, R.** *Expert One-on-One J2EE Design and Development.* s.l. : W. Press, 2003.

36. Extreme Programming: A gentle introduction. [En línea] http://www.extremeprogramming.org/.

37. Ejemplo de desarrollo software utilizando la metodología XP. [En línea] http://www.dsic.upv.es/asignaturas/facultad/lsi/ejemploxp/.

38. [En línea] http://wxdsgn.sourceforge.net/.

39. BloodshedSoftware. [En línea] http://www.bloodshed.net/devcpp.html.

40. [En línea] http://wxwidgets.org/.

41. Spring Security. [En línea] http://www.acegisecurity.org/.

42. Spring Framework. [En línea] http://www.springframework.org/.

43. Spring Source. [En línea] http://www.springsource.com/.

ANEXOS

Anexo I. Artefacto "Historia de usuario" propuesto por *eXtreme Programming (XP)*

Historia de Usuario	
Número:	**Nombre historia:**
Modificación (o extensión) de Historia de Usuario (Nro. y Nombre):	
Usuario:	**Iteración asignada:**
Prioridad en negocio: (Alta / Media / Baja)	**Puntos estimados:**
Riesgo en desarrollo: (Alta / Media / Baja)	**Puntos reales:**
Programador responsable:	
Descripción:	
Observaciones:	

Anexo II. Artefacto "Caso de prueba de aceptación" propuesto por *eXtreme Programming (XP)*

Caso de prueba de aceptación	
Código:	**Historia de Usuario:**
Nombre:	
Descripción:	
Condiciones de Ejecución:	
Entrada / Pasos de ejecución:	
Resultado Esperado:	
Evaluación de la Prueba:	

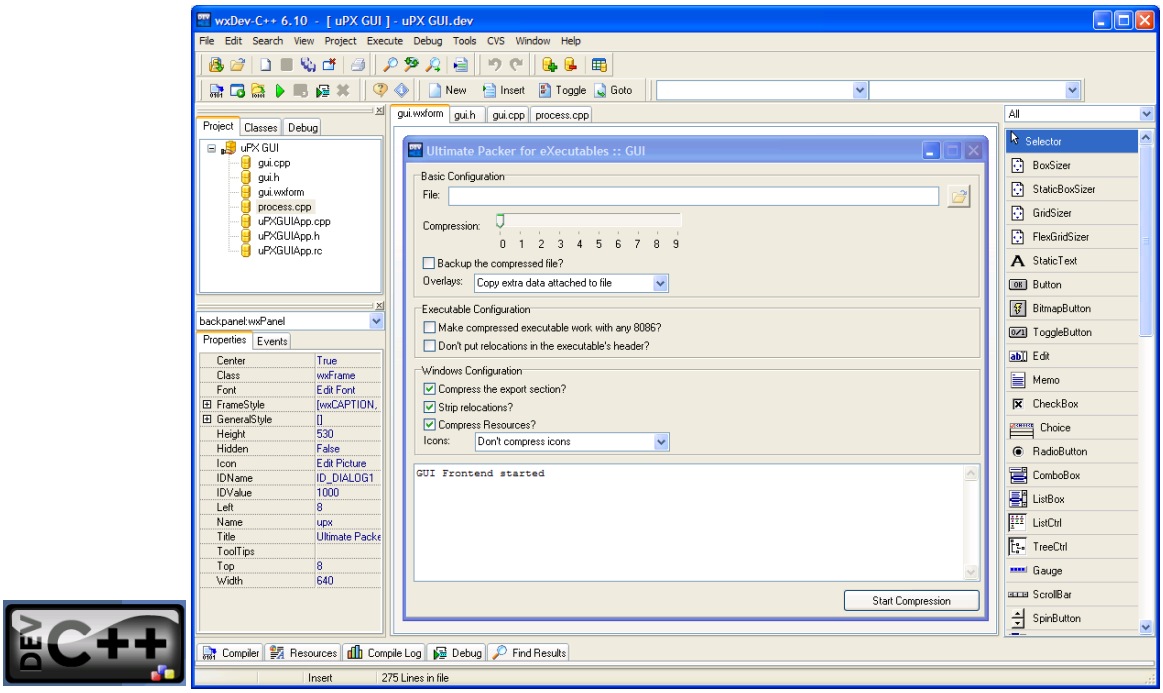
Anexo III. Artefacto "Tarea de ingeniería" propuesto por *eXtreme Programming (XP)*

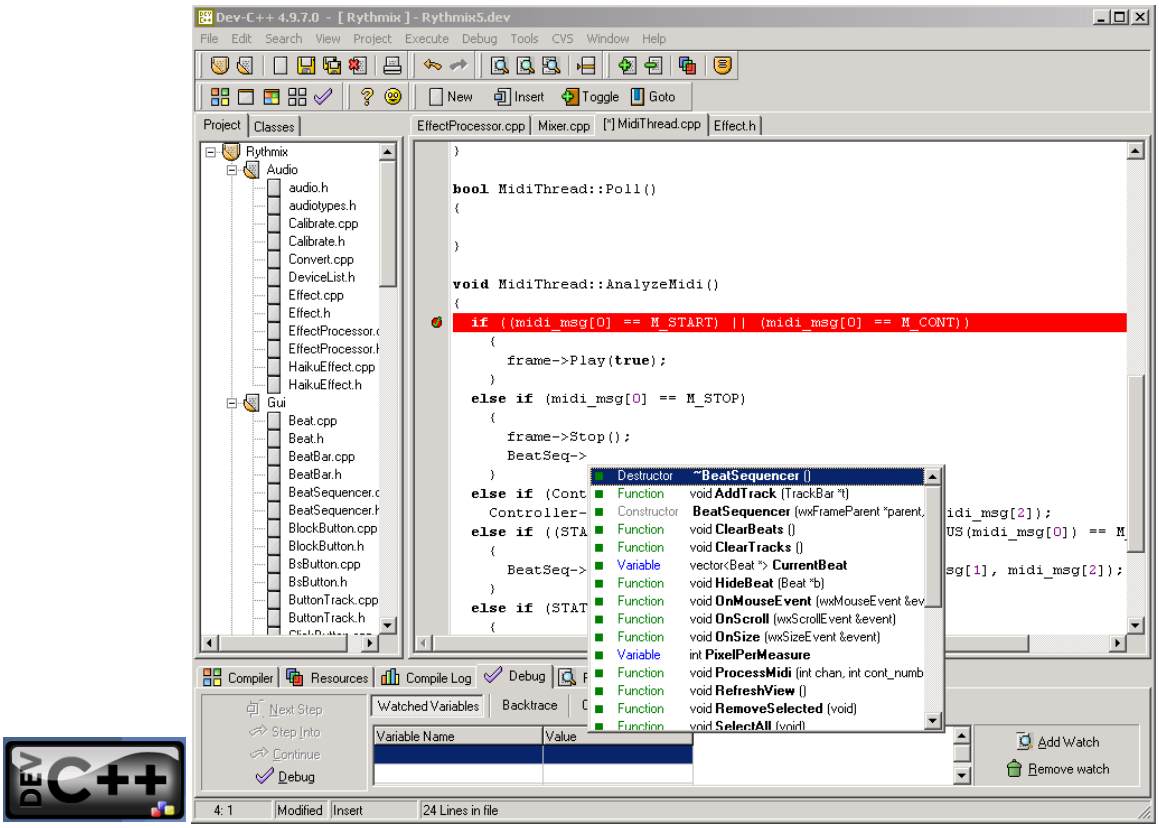
Tarea de ingeniería	
Número tarea:	**Número historia:**
Nombre tarea:	
Tipo de tarea:	**Puntos estimados:**
Fecha inicio:	**Fecha fin:**
Programador responsable:	
Descripción:	

Anexo IV. Artefacto "Tarjeta CRC" propuesto por *eXtreme Programming (XP)*

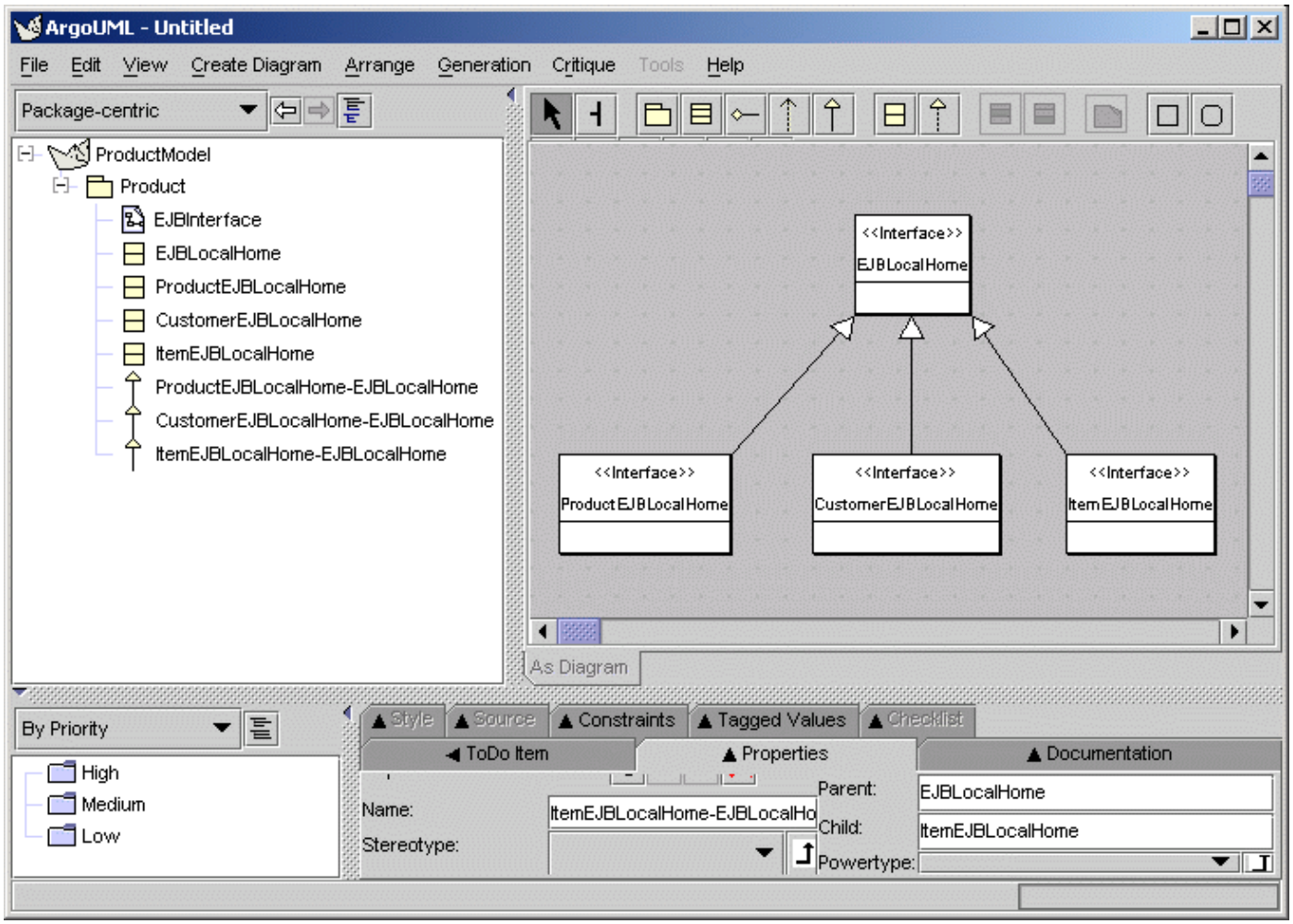
Nombre de la clase.	
Responsabilidades	Colaboradores

Anexo V. IDE wxDev-C++

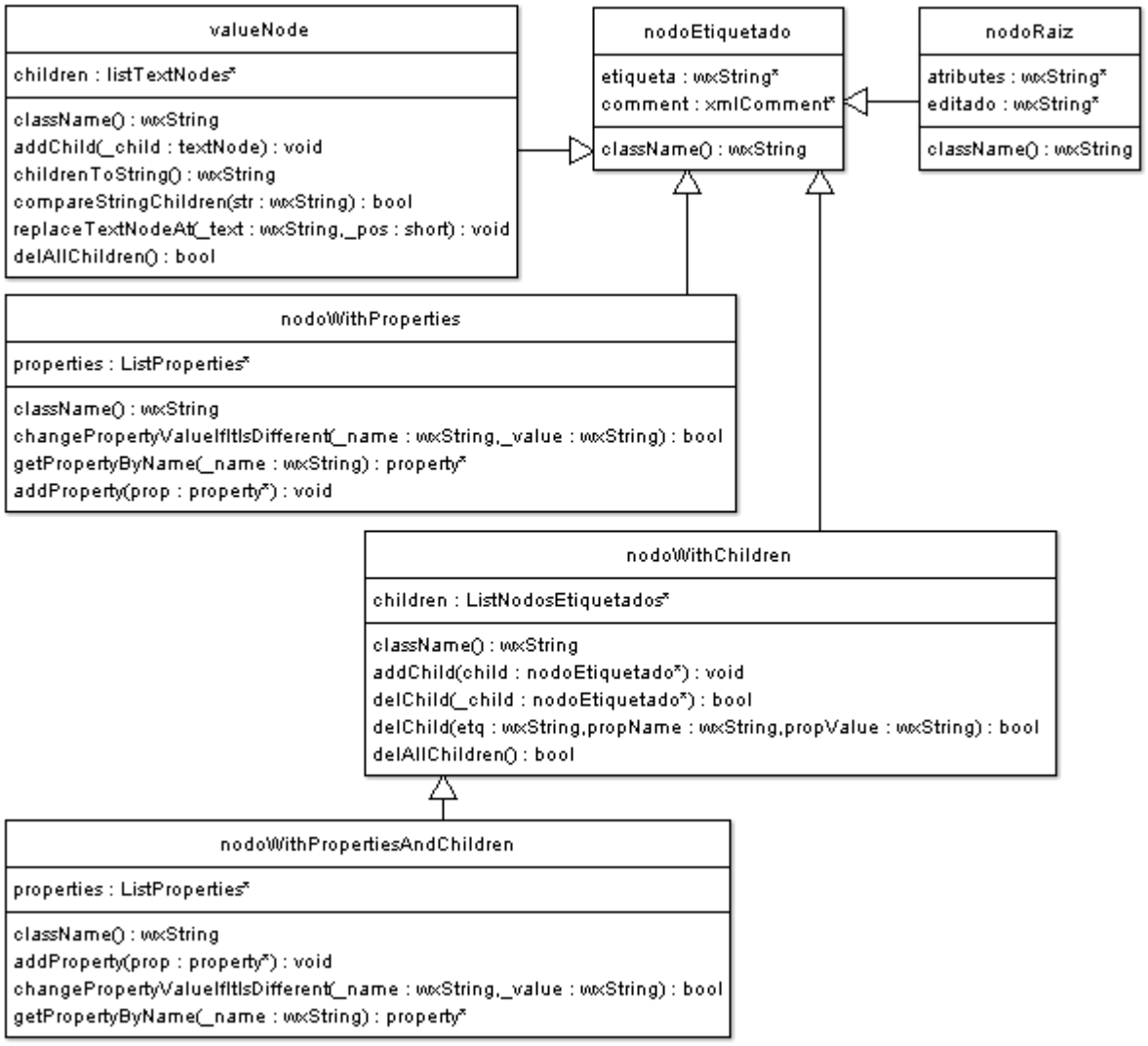




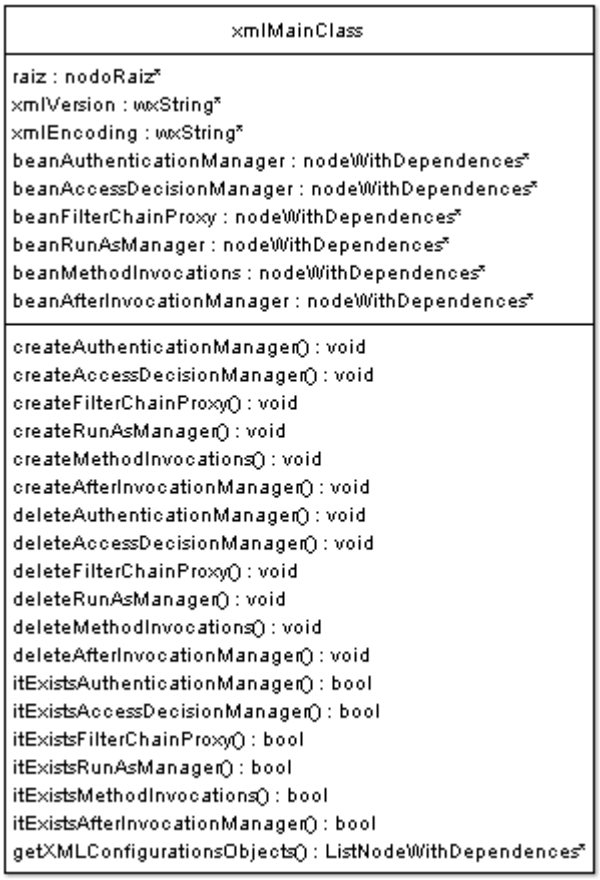
Anexo VI. ArgoUML

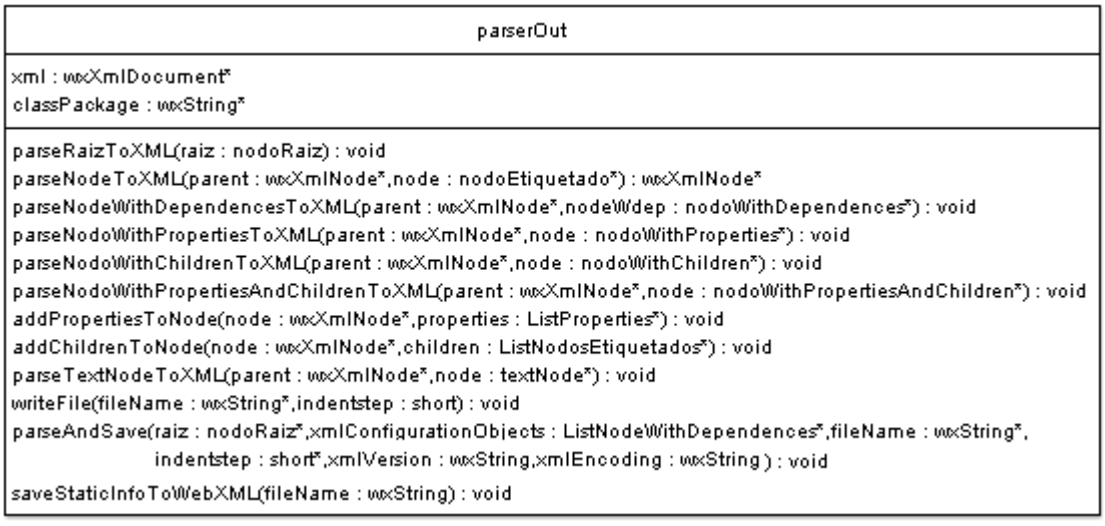


Anexo VII. Árbol de herencia de la súper-clase nodoEtiquetado



Anexo VIII. Clase controladora principal xmlMainClass, y Clase de acceso a datos parserOut





GLOSARIO DE TÉRMINOS

JEE	Java Platform, Enterprise Edition o Java EE (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación —parte de la plataforma Java— para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java con arquitectura de n niveles distribuida, basándose ampliamente en componentes de software modulares y ejecutándose sobre un servidor de aplicaciones. Permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores.
JSE	Java Platform, Standard Edition o Java SE (anteriormente conocido como Java 2 Platform, Standard Edition o J2SE hasta la versión 5.0), es una colección de APIs del lenguaje de programación Java útiles para muchos programas de esta plataforma. La Plataforma Java 2, Enterprise Edition incluye todas las clases en el Java SE, además de algunas de las cuales son útiles para programas que se ejecutan en servidores sobre workstations.
JME	Java Platform, Micro Edition o Java ME (anteriormente conocida como Java 2 Platform, Micro Edition o J2ME), es una colección de APIs de Java para el desarrollo de software para dispositivos de recursos limitados, como PDA, teléfonos móviles y otros aparatos de consumo.
XML	Del inglés Extensible Markup Language (Lenguaje de Marcas Extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.
API	Del inglés Application Programming Interface (Interfaz de Programación de Aplicaciones) es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

framework	En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.
PDA	Del inglés Personal Digital Assistant (Asistente Digital Personal), es un computador de mano originalmente diseñado como agenda electrónica (calendario, lista de contactos, bloc de notas y recordatorios) con un sistema de reconocimiento de escritura.
MVC	Modelo Vista Controlador, patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.
JSF	JavaServer Faces, framework para aplicaciones web en Java EE.
AOP	Del inglés Aspect Oriented Programming (Programación Orientada a Aspectos), es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.
EJB	Enterprise JavaBeans, son una de las API que forman parte del estándar de construcción de aplicaciones empresariales JEE de Sun Microsystems.
ORM	Del inglés Object/Relational Mapping (Mapeo Objeto/Relacional), es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.
JNDI	Interfaz de Nombrado y Directorio Java, es una Interfaz de Programación de Aplicaciones para servicios de directorio.
SSO	Single Sign-On, es un procedimiento de autenticación que habilita al usuario para acceder a varios sistemas con una sola instancia de identificación.

iBATIS	Framework de código abierto basado en capas desarrollado por Apache Software Foundation, que se ocupa de la capa de persistencia, se sitúa entre la lógica de negocio y la capa de la base de datos.
JSP	JavaServer Pages, es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.
JDBC	Acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.
AspectJ	Lenguaje de programación orientado por aspectos construido como una extensión del lenguaje Java creado en Xerox PARC.
wxWidgets	Bibliotecas multiplataforma y libres, para el desarrollo de interfaces gráficas programadas en lenguaje C++. Están publicadas bajo licencia LGPL, similar a la GPL con la excepción de que el código binario producido por el usuario a partir de ellas, puede ser propietario, permitiendo desarrollar aplicaciones empresariales sin coste. Proporcionan una interfaz gráfica basada en las bibliotecas ya existentes en el sistema (nativas), con lo que se integran de forma óptima y resultan muy portables entre distintos sistemas operativos. Están disponibles para Windows, MacOS, GTK+, Motif, OpenVMS y OS/2. También pueden ser utilizadas desde otros lenguajes de programación, aparte del C++: Java, Javascript, Perl, Python, Smalltalk, Ruby.
wxDEV-C++	Entorno de desarrollo integrado (IDE) libre basado en el popular Dev-C++. Tiene varias características nuevas no encontradas en el Dev-C++ original. Una de ellas es un diseñador RAD visual que trabaja como el C++Builder para crear aplicaciones wxWidgets. Soporta los compiladores MinGW y Visual C++ (6, 2003 y 2005). La última versión, la 6.10.2, agrega soporte para compiladores de Microsoft.
JAAS	Java Authentication and Authorization Service, pronunciado como "Jazz", es una Interfaz de Programación de Aplicaciones que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso.

DAO	En software de computadores, un Data Access Object (DAO, Objeto de Acceso a Datos) es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.
SHA	La familia SHA (Secure Hash Algorithm, Algoritmo de Hash Seguro) es un sistema de funciones hash criptográficas relacionadas de la Agencia de Seguridad Nacional de los Estados Unidos y publicadas por el National Institute of Standards and Technology (NIST).
MD5	En criptografía, MD5 (acrónimo de Message-Digest Algorithm 5, Algoritmo de Resumen del Mensaje 5) es un algoritmo de reducción criptográfico de 128 bits ampliamente usado.
X509	En criptografía, X.509 es un estándar UIT-T para infraestructuras de claves públicas (en inglés, Public Key Infrastructure o PKI). X.509 especifica, entre otras cosas, formatos estándar para certificados de claves públicas y un algoritmo de validación de la ruta de certificación.
Servlet	Objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. Un servlet es un objeto que se ejecuta en un servidor o contenedor JEE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente es HTML.
LDAP	LDAP (Lightweight Directory Access Protocol) es un protocolo a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red. LDAP también es considerado una base de datos (aunque su sistema de almacenamiento puede ser diferente) a la que pueden realizarse consultas.
RMI	RMI (Java Remote Method Invocation) es un mecanismo ofrecido en Java para invocar un método remotamente. Al ser RMI parte estándar del entorno de ejecución Java, usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java. Si se requiere comunicarse con otras tecnologías debe usarse CORBA o SOAP en lugar de RMI.