Universidad de las Ciencias Informáticas

Facultad 5

Desarrollo de un componente de seguridad para estandarizar el proceso de autorización en las aplicaciones informáticas desarrolladas en Java.

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

Autor: Arleis Prieto Riverón.

Tutor: Ing. Orestes Febles Díaz.

LA HABANA, 2012

DECLARACIÓN DE AUTORÍA

Declaro ser el autor del presente trabajo y autorizo a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo. Para que así conste, firmo la presente a los __ días del mes de ___ del año____.

Arleis Prieto Riverón

Firma del Autor

Ing. Orestes Febles Díaz

Firma del Tutor

DATOS DE CONTACTO

Nombre y apellidos: Orestes Febles Díaz.

Institución: Universidad de las Ciencias Informáticas (UCI).

Título: Ingeniero en Ciencias Informáticas.

e-mail: ofebles@uci.cu

AGRADECIMIENTOS

Quiero agradecer de forma especial a mis padres, porque gracias a ellos he llegado hasta aquí y a ellos les debo todo lo que soy. A mi hermana por quererme tanto y apoyarme siempre. A mi esposa por su amor incondicional, estar a mi lado en todo momento y por hacerme sentir la persona mas feliz del mundo. Quiero agradecer a mis abuelitas por todo su cariño y afecto.

Agradezco a el chino, Milagros, Sonia y Keylin por todo lo que han hecho por mi y por abrirme las puertas no solo de su casa sino también de su corazón.

Quiero agradecer a Miraida, Efren, Dayli y Dariel por ser los primeros en brindarme su apoyo cuando llegue aquí.

A mi tutor Orestes porque siempre fue una guía excelente y me ayudó en todo momento.

A Jorge Infante porque fue prácticamente un tutor más.

Agradezco a las muchachas del tribunal por sus críticas constructivas, sobre todo a yoisy que nunca perdió la paciencia conmigo.

Agradezco a Vita por ayudarme con el documento.

A mis amigos, compañeros y a todas las personas que de una forma u otra han contribuido con mi formación.

DEDICATORIA

Dedico este trabajo a la memoria de mi tío Rey, por todo lo que me enseñó y por lo importante que fue y continúa siendo en mi vida. A mis padres y a mi hermana por darme todo el amor y el cariño del mundo.

RESUMEN

En el presente trabajo se propone el desarrollo de un componente de seguridad que estandarice y externalice el proceso de autorización en las aplicaciones informáticas desarrolladas en Java. Para esto se analizan algunos de los principales conceptos relacionados con la seguridad en las aplicaciones. Se realiza un estudio del Lenguaje Extensible de Control de Acceso y de sus potencialidades para expresar políticas de autorización. Además se abordan algunas de las principales herramientas existentes que se encargan de la administración de la seguridad en las aplicaciones informáticas y que hacen uso del Lenguaje Extensible de Control de Acceso, arrojando como resultado el uso de la herramienta Servidor de Identidad. El componente consiste en una Interfaz de Aplicación Programable que permite la conexión y el consumo de los servicios expuestos en el Servidor de Identidad para tomar las decisiones de autorización, y en dependencia del resultado de estas, permitir o denegar el acceso a los recursos solicitados. El componente puede ser embebido dentro de aplicaciones desarrolladas en Java, pudiendo estas interactuar con el Servidor de Identidad, herramienta donde radican las políticas de autorización. A la solución desarrollada se le realizaron las pruebas de aceptación propuestas por la metodología de desarrollo utilizada, garantizando que los requisitos fueron cumplidos y que el sistema es aceptable.

Palabras claves: componente de seguridad, externalice el proceso de autorización, Lenguaje Extensible de Control de Acceso, Servidor de Identidad.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE figuras

INTRODUCCIÓN

Los sistemas informáticos tradicionales en un principio se organizaban en bloques monolíticos que encapsulaban la lógica del negocio y las funciones automatizadas. Dichos sistemas son conocidos como Cliente-Cliente, debido a que el servidor solo funciona como un almacén de información, mientras que la aplicación y el procesamiento de los datos son ejecutados en la máquina cliente. Esto trae como consecuencias que los sistemas monolíticos necesiten mayores requerimientos de hardware para las computadoras clientes, lentitud en el procesamiento de peticiones sencillas y mayor costo en el proceso de actualización. Además este tipo de aplicaciones ocupan mayor ancho de banda, provocando congestionamiento en la red local. [1]

Las desventajas de los sistemas descritos anteriormente, en conjunto con otros factores como la necesidad de asegurar la información, condicionaron la implementación de las aplicaciones Cliente-Servidor. Este tipo de aplicaciones están diseñadas de manera tal que separan la interfaz de usuario, la lógica del negocio y el acceso a datos en capas diferentes. La distribución en capas de los componentes de la aplicación permite un mejor rendimiento del sistema, debido a que no se ve la sobrecarga que existía sobre las máquinas clientes en las aplicaciones monolíticas. Este tipo de modelo tiene otras ventajas como la facilidad de mantenimiento y la escalabilidad. Sin embargo los sistemas distribuidos todavía no solucionan muchos de los problemas existentes, presentando desventajas como el empleo de mayor esfuerzo en el proceso de administración de cambios y mayor complejidad en el desarrollo de aplicaciones.

La evolución de las aplicaciones siguió su curso y surgieron las aplicaciones compuestas. Una aplicación compuesta es una colección de activos de software que se han ensamblado para ofrecer una capacidad de negocio. Estos activos son artefactos que se pueden implementar de manera independiente, permiten la composición y aprovechar las capacidades específicas de la plataforma.[2]

Las primeras aplicaciones compuestas se valieron de una arquitectura relativamente simple y estática, generalmente compuesta de una base de datos con un servidor de aplicaciones y una interfaz gráfica de usuario. Los sistemas informáticos de este tipo son usados en los más diversos entornos, pero es en el sector empresarial donde incursionan con más fuerzas debido a las ventajas que reporta su uso. Las aplicaciones compuestas se comunican entre sí, posibilitando que haya un flujo de información en tiempo real entre ellas, por esta razón son muy fáciles de actualizar, ya que este proceso ocurre de forma

dinámica. [3]

Todos los tipos de sistemas informáticos descritos anteriormente han logrado considerables mejoras en la productividad de las empresas y entidades, automatizando procesos de negocio que anteriormente se hacían de forma manual.

La seguridad es actualmente una de las apuestas más importantes de las empresas. Asegurar digitalmente todo lo que anteriormente se aseguraba en cajas fuertes o en papel, es el reto de la era digital y de la nueva sociedad de la información.[4] El uso de aplicaciones informáticas a la vez que introduce incalculables beneficios, trae consigo la aparición de un conjunto de vulnerabilidades de seguridad. Una vez que un determinado usuario está autenticado en un sistema, es necesario gestionar de forma eficiente su nivel de acceso a determinados recursos. La autorización constituye uno de los mecanismos de seguridad más importantes que debe tener implementado cualquier sistema.[5]

Aunque este proceso se ha perfeccionado en el transcurso de los años, en ocasiones no se realiza correctamente debido a que no se emplean estándares, ni se implementan soluciones lo más genéricas posibles, surgiendo así la siguiente **situación problémica**:

- Las aplicaciones administran su propia seguridad de forma independiente, con bases de datos y mecanismos que a veces son propios, lo que trae como consecuencia que la integración con otros sistemas sea engorrosa, debido a que la seguridad se implementa ligada a la lógica del negocio.
- No se realiza una administración centralizada de las políticas de seguridad, lo que trae como consecuencia que se dificulte controlar la ocurrencia de violaciones en los sistemas y la detección de las mismas.
- La construcción de aplicaciones conlleva al desarrollo reiterativo de un módulo de seguridad, cuyas funcionalidades en muchas ocasiones tienen gran semejanza, esto implica pérdida de tiempo y gastos innecesarios de recursos humanos y materiales.
- No existe una forma estandarizada para definir las políticas de autorización en las aplicaciones desarrolladas. Actualmente se usan las Listas de Control de Acceso (ACL, del inglés Access Control List) para este tema, lo que implica:
  - Distintos modelos de seguridad.
  - Distintos lenguajes para expresar las políticas.
  - Distintas interfaces para consultar y definir las ACLs.

Por tanto, se plantea el siguiente **problema a resolver:**

¿Cómo garantizar que el proceso de autorización en las aplicaciones informáticas desarrolladas en Java en la Universidad de las Ciencias Informáticas se realice de manera estandarizada y reutilizable?

Teniendo como **objeto de estudio** *la seguridad en el control de acceso en las aplicaciones informáticas desarrolladas en Java* y el **campo de acción** *el proceso de autorización en las aplicaciones informáticas desarrolladas en Java en la Universidad de las Ciencias Informáticas.*

Con el desarrollo de la presente investigación se persigue lograr como **objetivo general** *desarrollar un componente de seguridad para estandarizar y externalizar el proceso de autorización en las aplicaciones informáticas desarrolladas en Java en la Universidad de las Ciencias Informáticas.*

Para darle cumplimiento a este objetivo se plantearon los siguientes **objetivos específicos**:

- Analizar el estado del arte y elaborar el marco teórico referencial sobre el tema, así como definir las herramientas a utilizar en el desarrollo del componente de seguridad.
- Implementar el componente de seguridad.
- Probar el componente desarrollado.

Idea a defender: *Con el desarrollo de un componente de seguridad para estandarizar y externalizar el proceso de autorización en las aplicaciones informáticas desarrolladas en Java en la Universidad de las Ciencias Informáticas, se garantiza que este proceso se realice de manera estandarizada y reutilizable.*

Para lograr el alcance del objetivo se hace uso de diferentes **Métodos de Investigación**:

Métodos Teóricos:

Histórico Lógico: Se desarrolló un estudio del estado del arte de la problemática, se analizaron las características fundamentales de cada una de las herramientas y las tendencias en la resolución de esta problemática.

Analítico Sintético: Método que tiene como objetivo analizar la teoría, documentación y bibliografía que permitan extraer los elementos más importantes acerca del tema.

Inductivo Deductivo: Se utilizó para el planteamiento del objetivo y para realizar una propuesta de solución que se adapte a las necesidades que tienen las aplicaciones informáticas de manejar la autorización de forma independiente a la lógica del negocio.

Métodos empíricos:

Entrevista: Se realizaron diferentes entrevistas con el objetivo de obtener información detallada y profundizar en el tema con el fin de resolver el problema propuesto.

Encuestas: Se realizaron encuestas como técnica de adquisición de información usando un grupo de cuestionarios previamente elaborados.

Experimento: Para establecer las diferentes pruebas con el propósito de determinar la fiabilidad del sistema y el mejoramiento de la usabilidad del mismo, a través de la detección de errores.

Estructura del documento:

El presente trabajo de diploma está estructurado de la siguiente manera: resumen, introducción, tres capítulos de contenido, conclusiones, recomendaciones, referencias bibliográficas y por último los anexos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA: En este capítulo se analizan los principales conceptos y definiciones asociados al dominio del problema que son necesarios para la comprensión y el desarrollo de la investigación. Se hace un estudio de algunos de los lenguajes para expresar políticas de control de acceso. Se describen algunas de las principales herramientas existentes que se encargan de la administración de la seguridad en las aplicaciones informáticas. Además se hace un análisis de las tecnologías a utilizar, así como de la metodología de desarrollo de software seleccionada para el desarrollo del componente.

CAPÍTULO 2: PROPUESTA DEL SISTEMA. EXPLORACIÓN Y PLANIFICACIÓN:

En este capítulo se realiza la propuesta del componente a desarrollar. Durante la Fase de Exploración se describen las Historias de Usuarios (HU) en correspondencia con cada uno de los requisitos funcionales identificados. Se hace una descripción de todos los artefactos generados durante la fase de Exploración. Durante la Fase de Planificación se realiza la estimación del esfuerzo por cada HU, el plan de iteraciones y el plan de duración de cada iteración. Se define el plan de entregas con las propuestas de liberación de las versiones de la aplicación. También se realiza el diseño del componente mediante las tarjetas Clase-Responsabilidad-Colaboración (CRC).

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS: Durante la Fase de Implementación se describen las tareas de programación en correspondencia con cada una de las HU identificadas. También se hace una descripción de los estándares de codificación empleados en el desarrollo del componente. Durante la Fase de Pruebas se realizan las pruebas de aceptación y de rendimiento.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

En este capítulo se analizan los principales conceptos y definiciones asociados al dominio del problema que son necesarios para la comprensión y el desarrollo de la investigación. Se hace un estudio de algunos de los lenguajes para expresar políticas de control de acceso. Se describen algunas de las principales herramientas existentes que se encargan de la administración de la seguridad en las aplicaciones informáticas. Además se hace un análisis de las tecnologías a utilizar, así como de la metodología de desarrollo de software seleccionada para el desarrollo del componente.

1.2 CONCEPTOS ASOCIADOS AL DOMINIO DEL PROBLEMA

1.2.1 Control de acceso

Es el mecanismo que en función a la identificación ya autenticada permite acceder a datos o recursos. El control de acceso es el proceso de conceder permisos a usuarios o grupos de acceder a los recursos de un sistema. Está basado en tres conceptos fundamentales: identificación, autenticación y autorización. Los controles de accesos son necesarios para proteger la confidencialidad, integridad y disponibilidad de los objetos, y por extensión de la información que contienen, pues permiten que los usuarios autorizados accedan solo a los recursos que ellos requieren para realizar sus tareas.[6]

1.2.2 Autorización

La autorización es el procedimiento que permite determinar si el usuario o proceso previamente identificado y autenticado tiene permitido el acceso a los recursos. Autorización se refiere a conceder servicios específicos (entre los que se incluye la "negación de servicio") a un determinado usuario, basándose para ellos en su propia autenticación, los servicios que está solicitando, y el estado actual del sistema. Es posible configurar restricciones a la autorización de determinados servicios en función de aspectos como la hora del día, la localización del usuario, etc.[6]

1.2.3 Componente

Un componente es una parte no trivial, casi independiente, y reemplazable de un sistema, que llena claramente una funcionalidad dentro de un contexto en una arquitectura bien definida. Un componente se conforma y provee las funcionalidades por medio de un conjunto de interfaces bien definidas.[7]

1.3 LENGUAJES DE POLÍTICAS DE PRIVACIDAD Y CONTROL DE ACCESO

Existen varios lenguajes de privacidad con los cuales se pueden expresar las políticas definidas para ofrecer protección a la información y no exponerla innecesariamente. Entre ellos se pueden mencionar P3P, EPAL y XACML.

1.3.1 Listas de Control de Acceso

Las Listas de Control de Acceso (ACL) no son propiamente un lenguaje para expresar políticas de control de acceso, sino una estructura de datos organizada en torno a tres elementos fundamentales, sujeto (entidad interesada en obtener una autorización para realizar determinada acción), objeto (elemento sobre el cual la acción recae) y privilegios (acciones permitidas). La utilización de Listas de Control de Acceso fue introducida alrededor de 1970 con el sistema operativo Multics. Sus conceptos son utilizados actualmente en cualquier sistema que aplica controles de seguridad de acceso como sistemas operativos, servidores Web y de aplicaciones, bases de datos, etc. A pesar del difundido uso de este sistema y la simplicidad del proceso de autorización, no se ha propagado aún una forma de expresar las listas de control (o políticas de acceso) de manera portable y unificada. Esto ha determinado una proliferación de enfoques propietarios.[8]

1.3.2 Plataforma de Preferencias de Privacidad

La Plataforma de Preferencias de Privacidad (P3P) es un conjunto estandarizado de preguntas de opción múltiple, cubriendo los aspectos más importantes de las políticas de privacidad de los sitios web. Permite hacer accesible esta información de los sitios web en un formato estándar y legible para la máquina. También permite al explorador leer automáticamente la imagen del cuestionario y compararlo con el conjunto de preferencias de privacidad del usuario. Realza el control del usuario poniendo políticas de privacidad donde los usuarios pueden encontrarlas, en una forma que los usuarios pueden entender y lo más importante es que permite a los usuarios actuar en lo que ven.[9]

1.3.3 Lenguaje de Autorización y Privacidad Empresarial

El Lenguaje de Autorización y Privacidad Empresarial (EPAL) es un lenguaje interoperable para el intercambio de políticas de privacidad bajo un formato estructurado entre las aplicaciones de las empresas. EPAL fue diseñado para facilitar la traducción de las políticas de privacidad de la empresa en representaciones legibles para las máquinas

y poder procesar los datos necesarios para hacer cumplir las reglas de negocio de las empresas.[10]

1.3.4 Lenguaje Extensible de Control de Acceso

El Lenguaje Extensible de Control de Acceso (XACML) es un estándar de OASIS que describe el lenguaje de la política y el lenguaje de la decisión/respuesta de control de acceso (ambos escritos en XML). El lenguaje de la política es utilizado para describir en general los requerimientos del control de acceso y tiene puntos de extensiones estandarizadas para definir nuevas funciones, tipos de datos, lógica combinada, etc.[11] Este lenguaje de petición/respuesta permite realizar una consulta para preguntar si una acción debe ser permitida o no, interpretando el resultado obtenido. La respuesta siempre incluye uno de los siguientes valores: Permitir, Denegar, Indeterminado (ocurrió un error o hace falta un valor requerido, por lo que la decisión no pudo ser tomada) o No Aplicable (la petición no puede ser autorizada por este servicio).

El estándar XACML fue creado para establecer un esquema centralizado de nombres correspondientes a las expresiones de la autorización de políticas en XML de los objetos identificados. Al existir diversos lenguajes de políticas de control de acceso dentro de las aplicaciones, las políticas no son compatibles para ser utilizadas por diversas aplicaciones. La mayoría de los lenguajes actuales no soportan políticas distribuidas, no son extensibles o no tienen las suficientes expresiones para determinar nuevos requerimientos. XACML habilita el uso de atributos arbitrarios en las políticas de control de acceso basadas en roles, etiquetas de seguridad, políticas basadas en hora y/o fecha, políticas indexadas, políticas de negación de permisos, así como políticas dinámicas. Todo esto sin necesidad de realizar cambios en las aplicaciones que utilizan XACML. Está diseñado intencionalmente para ser utilizado en una amplia variedad de ambientes de aplicaciones. Las principales características de este lenguaje son[12]

- Definido en el esquema XML
- Establece esquemas para definir:
  - Políticas
  - Solicitud de Decisión de Acceso
  - Decisiones de Acceso
- Contiene un amplio conjunto de funciones y tipos de datos para expresar las políticas

XACML es una plataforma independiente basada en un lenguaje estandarizado de políticas de control de acceso. En el mismo se definen las reglas sobre cómo las decisiones de autorización parten de la evaluación de las políticas de control de acceso.[13] Ofrece una arquitectura que se ha diseñado para permitir a las organizaciones especificar las políticas de control de acceso de una forma simple y flexible. XACML es una herramienta poderosa a la hora de establecer políticas de control de acceso, procesamiento de solicitudes de acceso y manejo de decisiones de autorización. También define cómo es la estructura de intercambio de mensajes de autorización y un modelo para organizar y almacenar la información de autorización. Dicho modelo impone una estructura base pero con flexibilidad suficiente para que cada sistema exprese las políticas de autorización de la forma más conveniente a su dominio de discurso. [8]

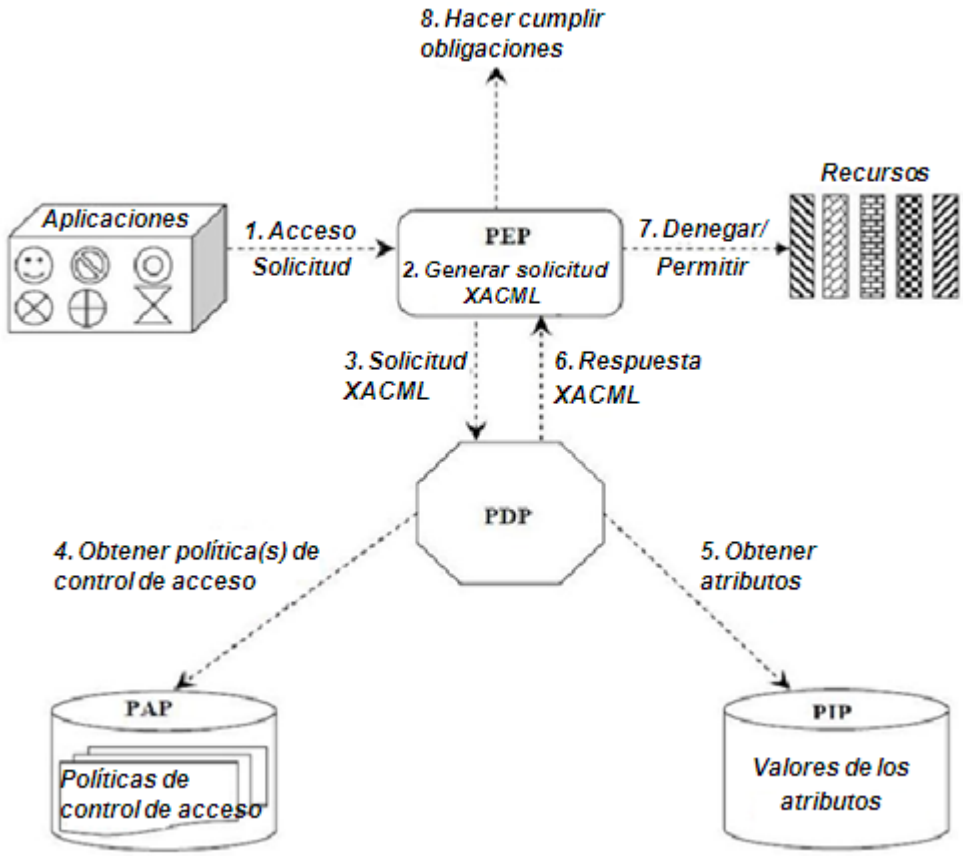


Figura 1. 1. Arquitectura XACML.

Los sistemas que implementen sus políticas de autorización basándose en XACML tendrán las siguientes ventajas.[14]

- Flexibilidad.
- Utilización de un lenguaje unificado y portable para expresar sus políticas.
- Extensibilidad.
- Reusabilidad.
- Facilitación del intercambio de políticas con otros sistemas.
- Escalabilidad.

El lenguaje XACML es el más flexible de los anteriormente mencionados, pues es el que puede adaptarse más fácilmente a cualquier aplicación que lo requiera, siempre y cuando los datos sean enviados en archivos XML. Debido a su flexibilidad y adaptación a cualquier tipo de aplicación se decidió utilizar el lenguaje XACML para la realización de este trabajo.

1.4 Principales elementos de XACML

XACML está formado por varios elementos, cada uno tiene una funcionalidad distinta para en conjunto poder realizar una evaluación exitosa de la solicitud de autorización en combinación con la política que se requiere. A continuación se definen los principales elementos[12]

- Punto de Administración de Política (PAP, Policy Administration Point) es el punto en el cual se crean y administran las políticas de control. Puede ser desde un editor XML de archivos de texto hasta un sistema encargado de encapsular un lenguaje de políticas propietario en la forma de un lenguaje XACML.
- Punto de Decisión de Política (PDP, Policy Decisión Point) es el punto que evalúa la Política aplicable y ejecuta una Decisión de Autorización.
- Punto de aplicación de Política (PEP, Policy Enforcement Point) es el punto que intercepta el pedido de autorización y lo deriva al PDP. Luego de obtener la respuesta del PDP, elabora una repuesta para el sistema que hizo el pedido de autorización.
- Punto de Información de Política (PIP, Policy Information Point) es el punto que actúa como una fuente de valores de atributos.

1.4.1 PEP

El PEP es un componente de la arquitectura de XACML que recibe las peticiones de acceso, extrae los atributos de la solicitud, dígase usuario, URL a la que está intentando acceder, entre otros, elabora una petición XACML y la envía al PDP para su

evaluación.[8] También se asegura de que todas las obligaciones con una decisión de autorización se ejecutan, entiéndase como obligación a una acción que debe realizarse junto con la aplicación de una decisión de autorización, y se especifica en una política de control de acceso.[13]

El PEP garantiza la ejecución de decisiones a través de las reglas definidas por el PDP. Generalmente el componente PEP se localiza dentro de la aplicación utilizada para proteger la información. Su arquitectura varía, puede ser un simple *if else*, un servicio o un filtro dentro de un XML, que intercepta en todos los casos, peticiones de acceso a servicios o aplicaciones.[15]

1.4.2 PDP

El Punto de Decisión de Política es el punto encargado de valorar una solicitud de autorización.[15] Según la información que el pedido contenga y el examen de las políticas de acceso existentes, determinará si el pedido debe ser rechazado o no.[8] El PDP recibe una solicitud XACML, obtiene las políticas aplicables del PAP, recupera los valores de los atributos del PEP, evalúa la solicitud en contra de las políticas de control de acceso y devuelve una decisión de autorización al PEP.[13]

1.4.3 PIP

El PIP es un componente que actúa como un directorio que almacena los valores de los atributos y los pone a disposición del PDP.[13] En algunos casos la evaluación de un pedido de autorización puede requerir la búsqueda de información de otras fuentes. Esta información se refiere concretamente a valores de determinados atributos. En estos casos, el pedido contiene información sobre el recurso que contiene al valor del atributo y el PIP es responsable de interpretar estos datos y obtener el valor.[8]

1.5 HERRAMIENTAS DE SEGURIDAD QUE IMPLEMENTAN XACML

Para la realización de este trabajo se necesita una herramienta que gestione las políticas de control de acceso XACML. A continuación se abordan algunas de ellas.

1.5.1 Oracle Entitlements Server

Es una herramienta desarrollada por Oracle que externaliza y centraliza la seguridad en las aplicaciones que la usen a través de políticas de grano fino. Permite a una organización la protección de sus recursos mediante la definición y gestión de las políticas XACML que controlan el acceso y uso de los mismos. Los privilegios de acceso se definen en una política especificando quién puede hacer qué con qué recursos, cuando se

puede hacer y cómo. Simplifica las políticas de autorización, y hace cumplir las decisiones de seguridad en aplicaciones distribuidas y heterogéneas. Oracle Entitlements Server asegura el acceso a los recursos de aplicaciones y componentes de software como direcciones URL, paginas JSP (Java Server Pages) y EJB (Enterprise JavaBeans), así como objetos arbitrarios de negocios como cuentas de clientes o registros de pacientes. Es una herramienta privativa. [16]

1.5.2 Axiomatics Policy Server

Es una herramienta de seguridad desarrollada por la compañía sueca Axiomatics que permite la gestión de políticas XACML. Permite a las empresas gestionar políticas de autorización portátiles y de grano fino. Es un sistema de control de acceso de gran alcance que permite a los usuarios administrar, simular y hacer cumplir las políticas de grano fino escritas en XACML. Axiomatics Policy Server fue la primera herramienta en soportar XACML 2.0 en su totalidad. En la versión 4.0 alcanzó una fortaleza tal que superó a todos sus competidores en una reciente evaluación muy completa de soluciones de autorización diferentes. Como resultado, el Axiomatics Policy Server se convirtió en el proveedor seleccionado para el despliegue más grande del mundo de la autorización basada en XACML, una solución construida para dar servicio a más de 200 millones de usuarios. Es una herramienta privativa.[17]

1.5.3 Tivoli Security Policy Manager

IBM Tivoli proporciona un catálogo completo de software de gestión de seguridad informática para reducir de forma rentable los riesgos y activos que dan soporte a procesos empresariales importantes. Dentro de este catálogo se encuentra la herramienta Tivoli Security Policy Manager que tiene entre sus objetivos fortalecer el control de acceso haciendo uso de XACML. Permite la gestión centralizada de las políticas de seguridad y agregar grano fino en el control de acceso para aplicaciones, bases de datos, portales y servicios. Tivoli Security Policy Manager permite cambiar y controlar las políticas de seguridad de forma centralizada, rápida, coherente y eficaz. Es una herramienta privativa.[18]

1.5.4 WSO2 Identity Server

WSO2 es una compañía que se dedica al desarrollo de aplicaciones de código abierto basado en SOA bajo la licencia Apache. El modelo de componentes de WSO2 permite un enfoque delgado, alto rendimiento, la auto-consistencia a través de la plataforma y

personalizar la adaptación de su proyecto. [19] WSO2 tiene clientes de diferentes tipos de industrias como ebay, BBC, Microsoft y HP.

Dentro del conjunto de herramientas que ofrece la suite de WSO2, se encuentra una que es la encargada de manejar todos los temas referentes a seguridad: El Identity Server. El cual empaqueta un proveedor de identidades, un motor XACML, una colección de componentes de fácil uso y una intuitiva consola de administración. Entre sus funcionalidades se pueden citar:

- Emisión de tokens de seguridad.
- Almacenamiento de políticas XACML.
- Soporte para OAuth.
- Single Sign On con SAML
- Autenticación.
- Autorización.
- PDP.

El uso de esta herramienta de la suite de WSO2 permite elaborar escenarios de seguridad más complejos como:

- Diseño e implementación de políticas XACML de grano fino.
- Diseño e implementación del escenario XMPP para garantizar una autenticación Open ID.
- Uso de Security Token Service para acceder a un servicio.
- Establecer la autenticación mutua basada en certificados digitales entre un cliente de un servicio web y el servicio web implementado en axis2.
- Ocultamiento de la contraseña en el escenario UT, ya sea pasando un hash o cifrando esa parte del mensaje.

La herramienta seleccionada para el desarrollo de este trabajo es el Identity Server. Es una de las más recomendables debido a la cantidad de funcionalidades que brinda y las muchas más que se le van incorporando, además de que es libre y de código abierto. Cuenta con un editor gráfico de políticas XACML y permite que las mismas sean importadas desde archivo. El Identity Server será la herramienta que hará función de servidor de políticas de autorización, la misma cuenta con un módulo para el diseño y posterior comprobación de políticas XACML, demostrando así que la propia herramienta

hará función de PDP. Esta herramienta tiene como aval positivo el haber alcanzado el reconocimiento internacional otorgado por la organización European Identity, debido a que en el 2011 el WSO2 Identity Server consiguió el premio a mejor solución informática en ofrecer implementaciones de XACML y OpenID. (Anexo 1)

En cuanto a rendimiento, el problema está solucionado debido a que el Identity Server incluye un mecanismo de cacheo de las decisiones, lo que reducirá notablemente los tiempos de respuesta y las transacciones por segundo. Todo podrá ser cacheado de manera tal que se evitarán bastantes consultas cuando se pida lo mismo. Las pruebas así lo confirman. (Anexo 2) Acerca de las preocupaciones por la alta disponibilidad y confianza se puede usar la siempre viable solución de clusterización, que viene por defecto en todas las soluciones de WSO2.

1.6 TECNOLOGÍAS ASOCIADAS AL DESARROLLO DEL SISTEMA

El desempeño de un proyecto está antecedido siempre por la investigación de las tecnologías que se utilizan, así como las ventajas y desventajas que trae su aplicación. Esta investigación fue realizada, analizando las tecnologías existentes que se ajustan a la solución a desarrollar. El componente debe estar libre de costos adicionales relativos a pago de licencias de software. A partir de la investigación realizada se identificaron las siguientes tecnologías a utilizar en el desarrollo del sistema.

1.6.1 Lenguaje de programación Java

Java es un lenguaje orientado a objetos. Es compilado, las clases que genera son interpretadas por la máquina virtual de Java. Siendo esta la que mantiene el control sobre las clases que se estén ejecutando. Es un lenguaje multiplataforma, el mismo código Java que funciona en un sistema operativo, funcionará en cualquier otro que tenga instalada la máquina virtual. La máquina virtual al ejecutar el código, realiza comprobaciones de seguridad, además el propio lenguaje carece de características inseguras, como por ejemplo los punteros.[20]

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el garbage collector (reciclador de memoria dinámica). Reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos.[21]

El lenguaje de programación seleccionado para el desarrollo de este trabajo es Java debido a su robustez ya que los errores se detectan en el momento en que se producen, lo que facilita la depuración. Además Java es muy seguro debido a que Applets recuperados por medio de la red no pueden causar daño a los usuarios. Por otra parte, este lenguaje de programación es orientado a objetos, motivo por el cual es muy beneficioso para el programador de aplicaciones.

1.6.2 Entorno de Desarrollo Integrado Eclipse

El entorno de desarrollo integrado Eclipse se caracteriza por presentar código abierto, por su portabilidad y también por ser multiplataforma. Este fue diseñado originalmente por la empresa IBM y actualmente es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Eclipse trabaja principalmente a base de módulos o *plugins*, lo cual hace posible el trabajo en variados lenguajes de programación como son Java, C/C++, PHP, Perl y Python. Esto lo diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.[22]

El IDE seleccionado para el desarrollo de este trabajo es el Eclipse por las ventajas que reporta su uso, además de que asegura robustez y rendimiento.

1.6.3 Lenguaje de modelado UML

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. UML es un lenguaje de modelado unificado que proporciona un medio gráfico para modelar varios componentes de un sistema de software.

UML ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. Debido a las características del problema a resolver y el componente a desarrollar, se ha decidido utilizar este lenguaje ya que es el más recomendable para el trabajo con lenguajes orientados a objetos.[23]

1.6.4 Herramientas CASE

Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora) son aplicaciones informáticas destinadas a aumentar la

productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como: realizar un diseño del proyecto, calcular costos, implementar parte del código automáticamente con el diseño dado, compilar automáticamente, documentar o detectar errores, entre otras. [24]

Entre los principales objetivos que se buscan al utilizar una herramienta CASE se tienen:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Reducir el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayudar a la reutilización del software, portabilidad y estandarización de la documentación
- Gestionar globalmente todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

1.6.4.1 Visual Paradigm.

Visual Paradigm es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. También proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML. Presenta licencia gratuita y comercial. Es fácil de instalar, actualizar y es compatible entre ediciones.[25]

Se decide utilizar esta herramienta CASE como medio para modelar el proyecto ya que tiene grandes ventajas entre las que se encuentran:

- Soporte de UML
- Modelado colaborativo con CVS y Subversion (control de versiones).
- Interoperabilidad con modelos UML2
- Ingeniería inversa - Código a modelo, código a diagrama.
- Diagramas de flujo de datos.
- Generador de informes.
- Licencia gratuita y comercial.
- Compatible entre ediciones

1.7 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Debido al desarrollo que ha alcanzado la producción de software a nivel mundial se ha creado una gran cantidad de documentación sobre las políticas, procesos y procedimientos para lograr la calidad del producto final.

Las metodologías de desarrollo de software se basan en una combinación de los modelos de proceso genéricos (cascada, evolutivo, incremental, etc.). Adicionalmente una metodología debería definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología al proyecto, guías para uso de herramientas de apoyo, etc.[26]

Según lo antes mencionado, una metodología puede definirse como un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software, ya que indica cómo obtener los distintos productos parciales y finales. Considerando su filosofía de desarrollo, estas se clasifican en metodologías tradicionales (no ágiles) y metodologías ágiles. Las metodologías tradicionales hacen mayor énfasis en la planificación y control del proyecto y en la especificación precisa de requisitos y modelado. Las metodologías ágiles están más orientadas a la generación de código con ciclos muy cortos de desarrollo, haciendo especial hincapié en aspectos humanos asociados al trabajo en equipo e involucrando activamente al cliente en el proceso.[27]

Entre los principales objetivos de las metodologías de desarrollo de software se encuentran cumplir con los requisitos iniciales del problema y minimizar las pérdidas de

tiempo en el proceso de desarrollo, así como minimizar riesgos e incrementar las posibilidades de éxito en el desarrollo de los productos.

1.7.1 Metodologías ágiles

Las metodologías ágiles son efectivas en proyectos con requisitos muy cambiantes, donde se exige reducir drásticamente los tiempos de desarrollo manteniendo una alta calidad de los mismos. Es una metodología sencilla, tanto en su aprendizaje como en su aplicación, lo que posibilita la reducción de los costos de implantación en un equipo de desarrollo. Están especialmente orientadas para proyectos pequeños y entre sus características más destacables está que dan un mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Entre las más usadas están SCRUM y Extreme Programming (XP).[28]

1.7.1.1 Extreme Programming (XP)

Constituye una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico. [28]

Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre. Entre las principales características de esta metodología se encuentran:

- Permite introducir nuevos requisitos o cambiar los anteriores ágilmente.
- Adecuado para proyectos pequeños y medianos.
- Adecuado para proyectos con alto riesgo.
- Su ciclo de vida es iterativo e incremental.
- Cada iteración dura entre una y tres semanas.
- No produce demasiada documentación acerca del diseño o la planificación.

XP se basa fundamentalmente en ser ligero, cercano al desarrollo, basado en Historias de Usuarios (HU), fuerte comunicación con el cliente, el código pertenece a todos, programación por parejas y pruebas como base de la funcionalidad. Las HU son

descompuestas en tareas de programación y asignadas a los programadores para ser implementadas durante una iteración.[28]

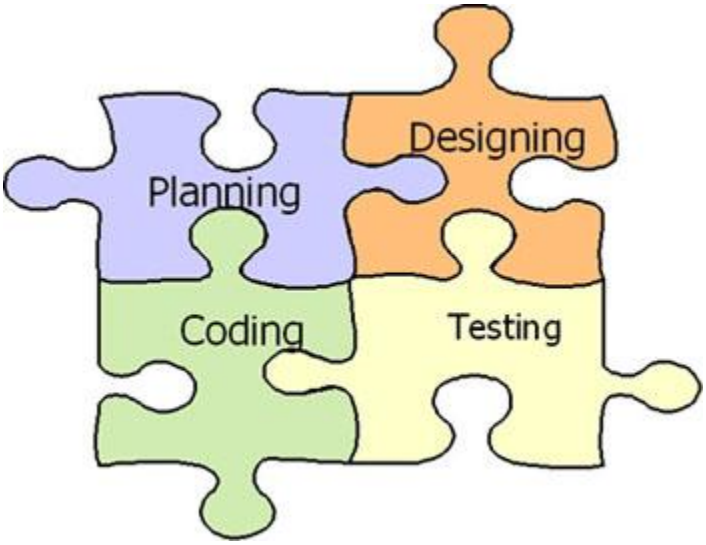


Figura 1. 2. Metodología Extreme Programming (XP).

Teniendo en cuenta las peculiaridades y características de las metodologías analizadas anteriormente y basándose en las particularidades del desarrollo del componente a implementar, se decidió utilizar XP como metodología de desarrollo de software, debido a que cuenta con características y ventajas que se adaptan a las necesidades actuales del desarrollo del componente. A continuación se mencionan algunas de ellas.

- Se consiguen productos usables con mayor rapidez.
- El proceso de integración es continuo, por lo que el esfuerzo final para la integración es nulo.
- Se atienden las necesidades del usuario con mayor exactitud. Esto se consigue gracias a las continuas versiones que se ofrecen al usuario.
- Se consiguen productos más fiables y robustos contra los fallos gracias al diseño de pruebas de forma previa a la codificación.
- Se consigue tener un equipo de desarrollo más contento y motivado, debido a que XP no permite excesos de trabajo (40 horas a la semana) y la comunicación entre los miembros del equipo consigue una mayor integración entre ellos.

- Que el software funcione es más importante que la documentación exhaustiva. La regla a seguir es: no producir documentos a menos que sean necesarios de forma inmediata. Estos documentos deben ser cortos y centrarse en lo fundamental.

1.8 CONCLUSIONES

Teniendo en cuenta el análisis hecho anteriormente, para darle solución al objetivo propuesto se seleccionó como metodología de desarrollo XP, una metodología ágil que permite gestionar los cambios de requisitos de forma rápida y es muy adecuada para proyectos pequeños. Como lenguaje de modelado se seleccionó UML, lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo del proyecto. Se propone también como herramienta CASE el Visual Paradigm ya que soporta UML y es multiplataforma.

Como lenguaje para expresar políticas de control de acceso se seleccionó XACML debido a que es un lenguaje estandarizado y altamente flexible, razón por la cual es de fácil adaptación a cualquier aplicación que lo requiera. La herramienta seleccionada como servidor de políticas XACML fue el Identity Server ya que es libre y de código abierto, además contiene un modulo para el diseño y posterior comprobación de las políticas, demostrando así que la propia herramienta hará función de PDP.

El IDE de desarrollo seleccionado fue el Eclipse porque es muy fácil de usar para la programación orientada a objetos, además de ser portable y multiplataforma. Como lenguaje de programación se escogió Java por ser el más apropiado para el desarrollo del componente.

CAPÍTULO 2: PROPUESTA DEL SISTEMA. EXPLORACIÓN Y PLANIFICACIÓN

2.1 INTRODUCCIÓN

En este capítulo se realiza la propuesta del componente a desarrollar. Durante la Fase de Exploración se describen las Historias de Usuarios (HU) en correspondencia con cada uno de los requisitos funcionales identificados. También se hace una descripción de todos los artefactos generados durante la fase de Exploración. Durante la Fase de Planificación se realiza la estimación del esfuerzo por cada HU, el plan de iteraciones y el plan de duración de cada iteración. Se define el plan de entregas con las propuestas de liberación de las versiones de la aplicación y se realiza el diseño del componente mediante las tarjetas Clase-Responsabilidad-Colaboración (CRC).

2.2 DESARROLLO DEL DIAGNÓSTICO

Las entidades así como las investigaciones sobre éstas, están sometidas a un cambio constante derivado de sus procesos de adaptación y las modificaciones del entorno. Por lo que requieren un diagnóstico para conocer su situación. La aplicación del diagnóstico incluye todas las actividades encaminadas a lograr una visión clara de la situación, de forma que se pueda determinar si realmente existe la necesidad de cambiar y, en caso de que así sea, hacia dónde deben orientarse los esfuerzos del cambio.[29]

Un diagnóstico es el conjunto de signos útiles para determinar el carácter peculiar de una enfermedad y/o no efectividad de la organización utilizado para cambiar o mejorar el estado de las cosas. Otros autores lo definen como: Proceso mediante el cual se lleva a cabo un análisis para buscar información que ayude a determinar la situación actual de la organización y detectar sus áreas de mejoramiento. También definido como: evaluación focalizada en un conjunto de variables que tienen relevancia central para la comprensión, predicción y control del comportamiento de un fenómeno determinado.[29]

En la presente investigación se realiza un diagnóstico con el objetivo de evaluar el nivel de conocimiento y uso de las políticas de autorización XACML en las aplicaciones informáticas desarrolladas en la UCI. El estudio diagnóstico se realizó a varios profesores con experiencia en el tema de seguridad en las aplicaciones informáticas. Como parte del mismo se realizaron búsquedas bibliográficas, encuestas y entrevistas a especialistas.

Para la obtención de la información se aplicaron diferentes métodos y técnicas de las identificadas en el diseño teórico metodológico de la investigación, éstos son:

- **Encuestas**: Se realizaron encuestas como técnica de adquisición de información usando un grupo de cuestionarios previamente elaborados, con lo que se conoció la valoración de varios profesores con respecto a temas relacionados con el objeto de estudio. (Anexo 3)
- **Entrevistas**: Fueron realizadas a varios especialistas con experiencias en los temas referentes en la seguridad.

A través de las encuestas realizadas se pudo comprobar que de un total de 10 profesores encuestados, el 70% conoce de la existencia de XACML como lenguaje para expresar políticas de autorización, aunque solo el 2% lo ha usado alguna vez. El 80% de los encuestados opinan que sería ventajoso externalizar la autorización en las aplicaciones, mientras que el 90% afirman que sería factible implementar un componente de seguridad para externalizar la autorización en las aplicaciones mediante el uso de políticas de autorización XACML.

2.3 PROPUESTA DEL SISTEMA

Se tiene como propósito desarrollar un componente de seguridad capaz de estandarizar y externalizar el proceso de autorización en las aplicaciones Java. Para esto se pretende hacer uso de XACML como lenguaje para expresar las políticas de autorización, así como el Identity Server de WSO2 como servidor de políticas y PDP. Dicho componente consiste en una biblioteca de clases empaquetada en un *jar*. La herramienta ofrece una Interfaz de Aplicación Programable (API, del inglés Application Programming Interface) que permite la conexión y el consumo de los servicios expuestos en el Identity Server para tomar las decisiones de autorización, y en dependencia del resultado de estas, permitir o denegar el acceso a los recursos solicitados. El componente puede ser embebido dentro de aplicaciones desarrolladas en Java, pudiendo estas interactuar con el Identity Server. Esto permitirá que las aplicaciones puedan externalizar el servicio de autorización, haciendo uso del estándar XACML .

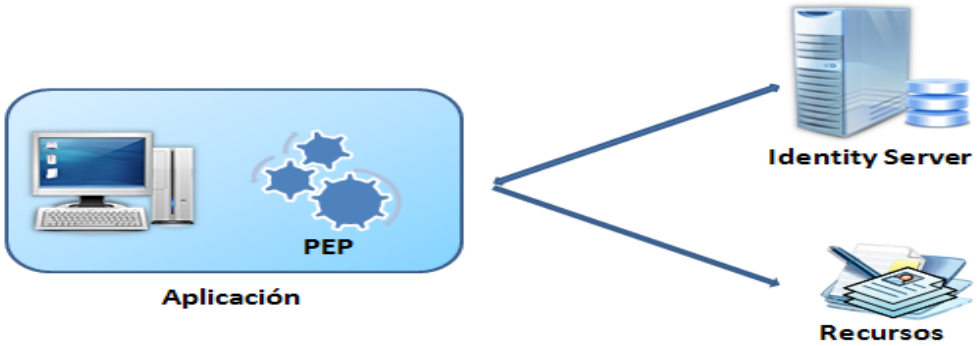


Figura 2. 1. Propuesta del sistema.

Esta herramienta permitirá a las aplicaciones conectarse al Identity Server mediante un servicio de autenticación que expone el mismo, llamado *Authentication Admin*. Esto se puede lograr configurando los datos de la conexión en una pequeña interfaz visual ofrecida por el componente que permite la entrada de los datos necesarios para llevar a cabo la conexión.

El componente permitirá la elaboración solicitudes de autorización XACML, el envío de las mismas al Identity Server y la obtención de la respuesta XACML correspondiente a cada petición realizada. Todo esto mediante el consumo del servicio Entitlement Service expuesto en el Identity Server.

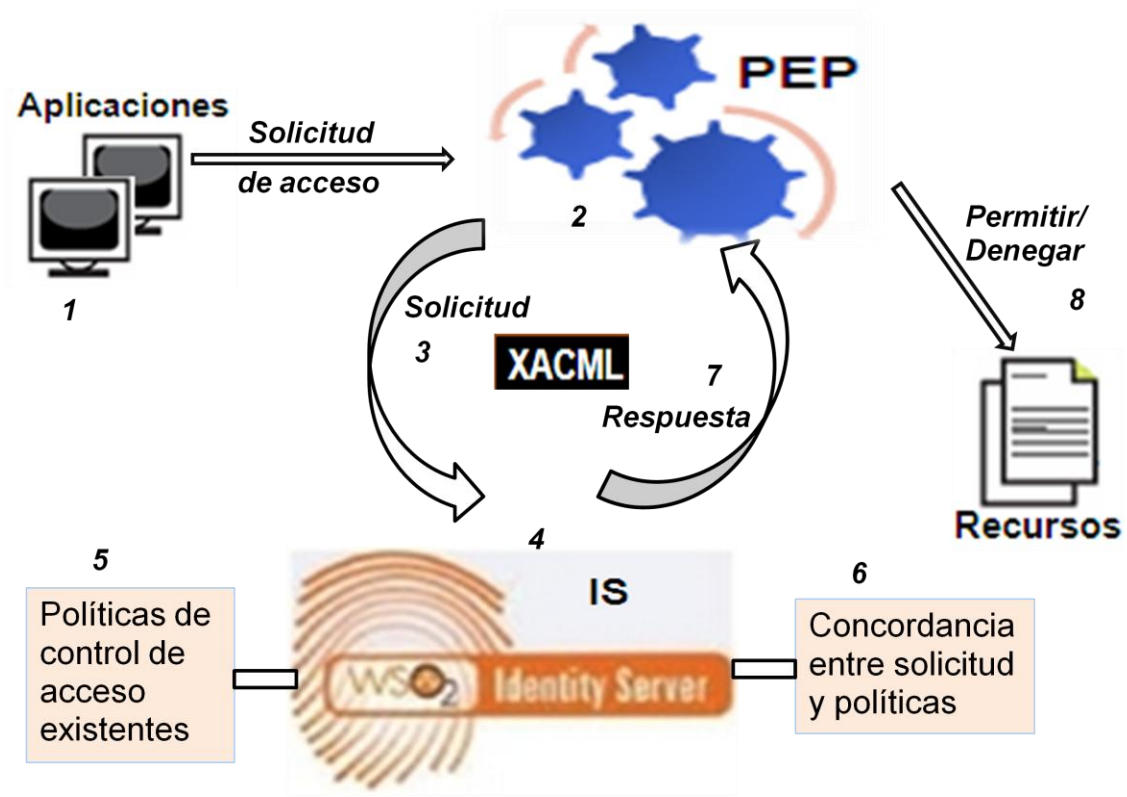


Figura 2. 2. Funcionamiento del componente PEP.

En (1) tenemos varias aplicaciones que solicitan acceso a determinado recurso de información. Esta solicitud es enviada (2) al componente PEP, donde la solicitud se transforma en un mensaje XACML (3) al que se le agrega información de identificación de quien hace la solicitud. Este mensaje XACML es enviado al PDP (4), en nuestro caso el Identity Server, el cual revisa si hay una concordancia (6) entre la información que contiene el mensaje y las políticas XACML (5) que tiene almacenada. En caso de encontrar una o varias concordancias aplica las políticas con las reglas contenidas que determinan si se responde (7) un Permit o Deny. Esta respuesta es devuelta hacia el componente PEP quien en función de la respuesta determina (8) si da acceso o no al recurso de información solicitado.

2.4 FASE DE EXPLORACIÓN

La metodología de desarrollo XP comienza con su fase de exploración. Durante esta fase se realiza el proceso de identificación de las HU. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el

proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración se lleva a cabo en pocas semanas o pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.[28]

2.4.1 Historias de Usuarios

Las HU son la forma en que se especifican en XP los requisitos del sistema. Estas se escriben desde la perspectiva del cliente, aunque los desarrolladores pueden brindar también su ayuda en la identificación de las mismas. El contenido de éstas debe ser concreto y sencillo. El tratamiento de las HU es muy dinámico y flexible, en cualquier momento pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o ser modificadas. Cada HU es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas pocas semanas.[30]

Respecto de la información contenida en la HU, existen varias plantillas sugeridas, pero no existe un consenso al respecto. En muchos casos sólo se propone utilizar un nombre y una descripción, o sólo una descripción, sumándole una estimación de esfuerzo en días.

Los contenidos de las fichas de las HU quedan estructurados de la siguiente forma:

Número: A cada HU se le asigna un número para facilitar su identificación por parte del equipo de desarrollo.

Nombre: Nombre descriptivo de la HU.

Prioridad en Negocio: Grado de prioridad que le asigna el cliente a la HU en dependencia de sus necesidades. Los valores que puede tomar son (Alta, Media o Baja).

Riesgo en Desarrollo: Grado de complejidad que le asigna el equipo de desarrollo a la HU luego de analizarla. (Alto, Medio o Bajo).

Puntos Estimados: Unidades de tiempo estimadas por el equipo de desarrollo para darle cumplimiento a la HU. Una unidad de tiempo equivale a una semana de trabajo de 40 horas.

Puntos Reales: Unidades de tiempo reales que el equipo de desarrollo necesitó para darle cumplimiento a la HU. Una unidad de tiempo equivale a una semana de trabajo de 40 horas.

Iteración Asignada: Número de la iteración en la cual será implementada la HU.

Descripción: Descripción simple sobre lo que debe hacer la funcionalidad a la que se hace referencia.

Durante la fase de exploración se identificaron cinco HU, las cuales se muestran a continuación.

HU 1: Guardar en un fichero los datos de conexión al PDP.

HU 2: Autenticar usuario contra PDP.

HU 3: Realizar solicitud de autorización a un recurso.

HU 4: Obtener decisión de autorización.

HU 5: Probar conexión al PDP.

HU	
Número: 1	**Nombre**: Guardar en un fichero los datos de conexión al PDP
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Puntos Estimados: 2 **Puntos Reales:** -	**Iteración Asignada:** 1
Descripción: Esta funcionalidad tiene como objetivo guardar en un fichero los datos de conexión al PDP entrados por el usuario.	

Tabla 2. 1 Historia de Usuario 1.

HU	
Número: 2	**Nombre**: Autenticar usuario contra el PDP
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto

Puntos Estimados: 3 **Puntos Reales:** -	**Iteración Asignada:** 1
Descripción: Esta funcionalidad tiene como objetivo autenticar a un determinado usuario contra el PDP.	

Tabla 2. 2 Historia de Usuario 2.

HU	
Número: 3	**Nombre**: Realizar solicitud de autorización a un recurso
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Puntos Estimados: 3 **Puntos Reales:** -	**Iteración Asignada:** 2
Descripción: Esta funcionalidad tiene como objetivo que un usuario realice una petición de autorización a un recurso determinado.	

Tabla 2. 3 Historia de Usuario 3.

HU	
Número: 4	**Nombre**: Obtener decisión de autorización
Prioridad en Negocio: Alta	**Riesgo en Desarrollo:** Alto
Puntos Estimados: 3 **Puntos Reales:** -	**Iteración Asignada:** 2
Descripción: Esta funcionalidad tiene como objetivo obtener la decisión de autorización a un recurso luego de haberse realizado una petición sobre el mismo.	

Tabla 2. 4 Historia de Usuario 4.

HU	
Número: 5	**Nombre**: Probar conexión al PDP
Prioridad en Negocio: Media	**Riesgo en Desarrollo:** Medio
Puntos Estimados: 1 **Puntos Reales:** -	**Iteración Asignada:** 3
Descripción: Esta funcionalidad tiene como objetivo probar conexión al PDP con la intención de saber si el mismo se encuentra funcionando.	

Tabla 2. 5 Historia de Usuario 5.

2.5 FASE DE PLANIFICACIÓN

En esta fase el cliente establece la prioridad de cada HU, y en correspondencia, los desarrolladores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Las estimaciones de esfuerzo asociado a la implementación de las HU la establecen los desarrolladores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de uno a tres puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la rapidez a la que se desarrolla, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las HU que fueron terminadas en la última iteración.[30]

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del

sistema, se divide la suma de puntos de las HU seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.[30]

2.5.1 Estimación de esfuerzo por Historias de Usuarios

Teniendo en cuenta la prioridad que tiene una determinada HU en el desarrollo del componente se decide en que iteración será implementada. Las HU que cuentan con mayor importancia por ser funcionalidades indispensables para la aplicación deben ser implementadas en las primeras iteraciones del ciclo de desarrollo.

A continuación se muestra mediante una tabla la planificación de las diferentes HU para cada iteración teniendo en cuenta su prioridad.

No	HU	Prioridad	Puntos de Estimación
1	Guardar en un fichero los datos de conexión al PDP	Alta	2
2	Autenticar usuario contra el PDP	Alta	3
3	Realizar solicitud de autorización a un recurso	Alta	3
4	Obtener decisión de autorización	Alta	3
5	Probar conexión al PDP	Media	1

Tabla 2. 6 Estimación de esfuerzo por Historias de Usuarios.

El tiempo total estimado para el desarrollo del sistema es de 12 puntos de estimación, que equivalen a 3 meses ideales de trabajo.

2.5.2 Plan de Iteraciones

Una vez identificadas las HU y estimado el esfuerzo propuesto para el desarrollo del sistema de cada una de estas historias, se procede a la planificación de la etapa de

implementación del sistema. Teniendo en cuenta la prioridad que tiene una determinada HU en el desarrollo del componente se decide en que iteración será implementada. Las HU que cuentan con mayor importancia por ser funcionalidades indispensables para la aplicación deben ser implementadas en las primeras iteraciones del ciclo de desarrollo. Se decide realizar el sistema en tres iteraciones, a continuación se muestran cuales son.

2.5.2.1 Iteración 1

Esta iteración tiene como objetivo la implementación de algunas de las HU de mayor prioridad. Durante el transcurso de la iteración se creará la base de la arquitectura del componente logrando cierta funcionalidad. Al finalizar se contará con las funcionalidades descritas en las HU 1 y 2 que son Guardar en un fichero los datos de conexión al PDP y Autenticar usuario contra el PDP.

2.5.2.2 Iteración 2

El objetivo de esta iteración es la implementación de la funcionalidad descrita en la HU 3 que lleva por nombre Realizar solicitud de autorización a un recurso, además de la funcionalidad que se describe en la HU 4 nombrada Obtener decisión de autorización. Al finalizar la iteración se contará con una versión de prueba de las funcionalidades implementadas, además de las implementadas en la iteración anterior.

2.5.2.3 Iteración 3

En esta iteración serán implementadas las funcionalidades de prioridad media. Esta está descrita en la HU 5 que lleva por nombre Probar conexión al PDP. Como resultado de esta iteración se tendrá la versión 1.0 del producto final. A partir de este momento el componente será puesto a prueba por un período de tiempo para evaluar el desempeño del mismo.

2.5.3 Plan de duración de las iteraciones

En el ciclo de vida de un proyecto regido por la Metodología XP se crea el plan de duración de cada una de las iteraciones, en este caso se hace para el único equipo de desarrollo con el cual se cuenta. Este plan tiene como finalidad mostrar el tiempo de duración de cada iteración, así como el orden en que serán implementadas las HU en cada una de las mismas.

No	Iteración	Orden de las HU a implementar	Duración total de la iteración
1	Iteración 1	Guardar en un fichero los datos de conexión al PDP Autenticar usuario contra el PDP	5 semanas
2	Iteración 2	Realizar solicitud de autorización a un recurso Obtener decisión de autorización	6 semanas
3	Iteración 3	Probar conexión al PDP	1 semana

Tabla 2. 7 Plan de duración de las iteraciones.

2.5.4 Plan de entregas

A continuación se propone el Plan de entregas diseñado para la fase de implementación, donde se hace una propuesta de la fecha aproximada en que se realizarán versiones al sistema al finalizar cada iteración en la fase de implementación.

Entregable	Final 1ra iteración 3ra semana de abril	Final 2da iteración 1ra semana de mayo	Final 3ra iteración 4ta semana de mayo
Componente de seguridad Punto de Aplicación de Política (PEP)	Versión 0.1	Versión 0.2	Versión 1.0

Tabla 2. 8 Plan de entregas.

2.6 ARQUITECTURA DEL COMPONENTE

Para el desarrollo del componente se propone usar arquitectura de N Capas, pues permite organizar el modelo de diseño de forma que las capas puedan estar físicamente distribuidas, lo que significa que los componentes de una capa sólo pueden hacer referencia a componentes en capas inmediatamente inferiores. Esta arquitectura reduce las dependencias entre capas de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores, lo que permite simplificar la comprensión y la organización de sistemas complejos. Para la aplicación se definieron tres capas. La capa de presentación, compuesta por una interfaz visual que le permite al usuario la entrada de los datos de conexión al PDP. La capa de lógica del negocio, se manifiesta a través de un grupo de clases encargadas de manejar el envío y recepción de solicitudes y decisiones de autorización respectivamente. La capa de acceso a datos, presente a través de las clases encargadas de guardar y leer en el fichero de configuración de los datos de conexión la PDP.

2.7 DISEÑO DEL COMPONENTE

El diseño adecuado para el software es aquel que supera con éxito todas las pruebas, no tiene lógica duplicada, refleja claramente la intención de implementación del equipo de desarrollo y tiene el menor número posible de clases y métodos. Para el diseño de las aplicaciones, la metodología XP no requiere la representación del sistema mediante diagramas de clases utilizando notación UML. En su lugar se usan otras técnicas como las tarjetas CRC (contenido, responsabilidad y colaboración) como una extensión informal a UML.[30] No obstante el uso de estos diagramas puede aplicarse siempre y cuando influyan en el mejoramiento de la comunicación, no sea un peso su mantenimiento, no sean extensos y se enfoquen en la información importante. (Anexo 3)

La técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis guiados por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades. Es importante resaltar que esta tarea es permanente durante la vida del proyecto partiendo de un diseño inicial que va siendo corregido y mejorado en el transcurso de este.[31]

2.7.1 Tarjetas CRC (Clase, Responsabilidades y Colaboración)

Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos, además de posibilitar que el equipo

completo contribuya en la tarea del diseño. Fueron propuestas por Ward Cunningham y Kent Beck. Se utilizan normalmente cuando se determinan las clases que se necesitan y cómo van a interactuar, luego se implementa la solución. Una tarjeta CRC representa un objeto. El nombre de la clase se coloca a modo de título en la tarjeta, las responsabilidades se colocan a la izquierda, y las clases que se implican en cada responsabilidad a la derecha, en la misma línea que su requerimiento correspondiente, tal y como muestra a continuación.

Clase	
Responsabilidades	**Colaboradores**

Tabla 2. 9 Tarjetas CRC.

Estas tarjetas CRC se hacen con el objetivo de identificar jerarquías de generalización/especificación, o jerarquías de agregación/composición entre las clases, de manera que ayuda al refinamiento de clases. Se definen además con la finalidad de obtener un diseño simple y no incurrir en la implementación de características que no son necesarias.

Clase: Nombre de la clase con que se está modelando.
Responsabilidades: Las responsabilidades de una clase son las cosas que conoce y las que realizan, sus atributos y métodos.
Colaboradores: los colaboradores de una clase son las demás clases con las que trabaja en conjunto para llevar a cabo sus responsabilidades.

PEP	
Descripción: Interfaz que expone las funcionalidades del componente	
Responsabilidades	
Nombre	**Colaboradores**
Probar conexión al PDP Autenticar usuario	

Iniciar Entitlement Elaborar petición de autorización a un recurso Obtener decisión de autorización	

Tabla 2. 10 Tarjetas CRC de la interfaz PEP.

Fichero	
Descripción: Clase encargada de manejar el fichero de datos de configuración del componente	
Atributos	
Nombre	**Descripción**
ficheroW ficheroR lectorFichero dirección nombreFichero	Atributo de tipo FileWriter Atributo de tipo FileReader Atributo de tipo BufferedReader Dirección donde se guarda el fichero de configuración Atributo que almacena el nombre del fichero de configuración
Responsabilidades	
Nombre	**Colaboradores**
Salvar fichero Leer fichero	

Tabla 2. 11 Tarjetas CRC de la clase Fichero.

ComponentePrincipal	
Descripción: Clase principal	
Atributos	
Nombre	**Descripción**
entstub	Atributo de tipo EntitlementServiceStub
authstub	Atributo de tipo AuthenticationAdminStub
authCookie	Atributo encargado de almacenar la cookie de autorización
urlServidor	Atributo que almacena la dirección URL del PDP.
direccionCertificado	Atributo que contiene la dirección del certificado
claveCertificado	Atributo que contiene la dirección del certificado
usuario	Atributo que contiene el usuario que intenta acceder a un recurso
Responsabilidades	
Nombre	**Colaboradores**
Probar conexión al PDP	AuthenticationAdminStub, Fichero
Autenticar usuario	AuthenticationAdminStub, Fichero
Iniciar Entitlement	EntitlementServiceStub, Fichero
Elaborar petición de autorización a un recurso	EntitlementServiceStub
Obtener decisión de autorización	EntitlementServiceStub

Tabla 2. 12 Tarjetas CRC de la clase ClienteEntitlement.

DeterminarSO	
Descripción: Clase que permite determinar el sistema operativo sobre el cual corre la aplicación	
Responsabilidades	
Nombre	**Colaboradores**
Determinar el sistema operativo sobre el cual corre la aplicación	

Tabla 2. 13 Tarjetas CRC de la clase DeterminarSO.

JkSFiltro	
Descripción: Clase que permite filtrar los archivos por la extensión jks	
Atributos	
Nombre	**Descripción**
jks	Atributo que contiene el nombre de la extensión por la cual se hace el filtro
Responsabilidades	
Nombre	**Colaboradores**
Filtrar los archivos por la extensión jks	

Tabla 2. 14 Tarjetas CRC de la clase JkSFiltro.

ComponenteApp
Descripción: Clase visual que permite la entrada de los datos que se guardan en el fichero de configuración
Responsabilidades

Nombre	Colaboradores
Permitir visualmente la entrada del los datos de configuración del componente	JkSFiltro
Guardar los datos en el fichero de configuración	Fichero

Tabla 2. 15 Tarjetas CRC de la clase ComponenteApp.

AuthenticationAdminStub	
Descripción: Clase generada a partir del wsdl del servicio AuthenticationAdmin	
Responsabilidades	
Nombre	**Colaboradores**
Permitir la implementación de clientes a partir de ella	AuthenticationAdminCallbackHandler, AuthenticationAdminAuthenticationExce-ptionException

Tabla 2. 16 Tarjetas CRC de la clase AuthenticationAdminStub.

EntitlementServiceStub	
Descripción: Clase generada a partir del wsdl del servicio EntitlementService	
Responsabilidades	
Nombre	**Colaboradores**
Permitir la implementación de clientes a partir de ella	EntitlementServiceCallbackHandler, EntitlementServiceException

Tabla 2. 17 Tarjetas CRC de la clase EntitlementServiceStub.

2.7.2 Patrones de diseño

Después de analizar algunos de los patrones de diseño más empleados para determinar si sería efectivo realmente su uso en cuanto a reusabilidad, extensibilidad y mantenimiento que se necesita en la solución desarrollada. Se seleccionaron los siguientes:

El patrón estructural **Fachada** (*Facade*) define una interfaz que permite acceder solamente a las funcionalidades que esta posee de forma sencilla y clara por parte de los clientes ocultando la complejidad del sistema. Favorece un acoplamiento débil entre el sistema y sus clientes, ayuda a dividir un sistema en capas y reduce dependencias de compilación. Por las características anteriormente mencionadas se decidió aplicarlo en la solución.

Patrones de Diseño GRASP

Los patrones GRASP (Patrones Generales de Asignación de Responsabilidades) son patrones de diseño que se usan para asignar responsabilidades a una clase. Se pueden destacar 5 patrones básicos (principales) y 4 patrones adicionales a aplicar en el diseño Orientado a Objetos. A continuación se muestran los más utilizados en la solución propuesta, con una breve descripción y ejemplos donde son aplicados.

- *Experto*: Asignar una responsabilidad al más competente en información, la clase cuenta con la información necesaria para cumplir la responsabilidad. Es el principio básico de asignación de responsabilidades que suele utilizarse en el diseño Orientado a Objetos. Es el patrón que más se usa para asignar responsabilidades.

  Utilización: Este patrón se puede observar en cada una de las clases pertenecientes a la capa de acceso a datos.

- *Controlador*: Asignar la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Asigna la responsabilidad del manejo de mensajes de los eventos de un sistema a una clase.

  Utilización: Este patrón es usado en la clase ComponenteApp perteneciente a la capa de presentación, la cual es la encargada de manejar los eventos generados por el usuario sobre la interfaz. Por ejemplo: cuando se guardan los datos de conexión la PDP.

2.8 CONCLUSIONES

Durante la elaboración de este capítulo se hace referencia a las fases de Exploración y Planificación propias de la metodología de desarrollo utilizada para la implementación de la aplicación. Se asumió una implementación por etapas la cual fue concebida y debidamente detallada. Durante la Fase de Exploración se describieron las HU en correspondencia con cada uno de los requisitos funcionales identificados. También se hace una descripción de todos los artefactos generados mediante las HU. Durante la Fase de Planificación se realizó la estimación del esfuerzo por cada HU que permitió tener un aproximado del tiempo que tomará la etapa de Implementación. Además se realizó el plan de iteraciones y el plan de duración de cada iteración. También se definió un plan de entregas con las propuestas de liberación de las versiones de la aplicación. Se introdujeron además, para apoyar el proceso de implementación, las tarjetas CRC para la representación de las clases. Por último se analizaron conceptos relacionados con los patrones de diseño y arquitectura utilizados para la futura implementación del componente.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

3.1 INTRODUCCIÓN

En el presente capítulo se describen las fases de implementación y pruebas. Se detallan las tareas de programación pertenecientes a las iteraciones llevadas a cabo durante la etapa de codificación del componente. Además se hace alusión a la fase de prueba, propia de la metodología de desarrollo utilizada para el desarrollo de la aplicación. También se detallan las pruebas de aceptación realizadas sobre el componente y se hace una pruebas de rendimiento.

3.2 FASE DE IMPLEMENTACIÓN

La implementación o codificación es el flujo de trabajo donde se producen los componentes del sistema: ejecutables, ficheros de código fuente, scripts, entre otros. Tiene como objetivo principal desarrollar la arquitectura y el sistema como un todo, así como definir la organización del código. La Metodología XP plantea que la implementación de un software debe realizarse de forma iterativa, obteniendo al culminar cada iteración un producto funcional que debe ser probado y mostrado al cliente para incrementar la visión de los desarrolladores con la opinión de éste. [31]

En esta fase se genera todo el código fuente necesario para satisfacer las HU definidas para la solución y se describen todas las tareas realizadas en cada iteración. Al inicio de cada HU, se lleva a cabo una revisión del plan de iteraciones y se modifica de ser necesario. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador o grupo de programadores como responsables. Estas tareas, pueden escribirse utilizando un lenguaje técnico y no necesariamente deben ser entendibles para el cliente.[30]

Para la implementación del componente se determinaron en la fase de planificación tres iteraciones de desarrollo. A continuación se hace referencia a los principales aspectos de las tareas de programación realizadas en las distintas iteraciones.

No	HU	Iteración	Tareas de programación

			No	Nombre	Puntos estimados
1	Guardar en un fichero los datos de conexión al PDP	1	1	Crear estructura inicial del componente usando las herramientas propuestas	0.5
			2	Implementar la clase Fichero	0.5
			3	Diseñar interfaz visual para la configuración de los datos de conexión al PDP	0.5
			4	Escribir pruebas para la HU 1	0.5

Tabla 3. 1 Tareas de programación de la HU 1.

No	HU	Iteración	Tareas de programación		
			No	Nombre	Puntos estimados
2	Autenticar usuario contra el PDP	1	5	Generación del cliente del servicio AuthenticationAdmin	1
			6	Implementar la funcionalidad autenticar usuario	1
			7	Escribir pruebas para la HU 2	1

Tabla 3. 2 Tareas de programación de la HU 2.

No	HU	Iteración	Tareas de programación		
			No	Nombre	Puntos estimados
3	Realizar solicitud de autorización a un recurso	2	8	Generación del cliente del servicio EntitlementService	1
			9	Implementar la funcionalidad realizar solicitud de autorización	1
			10	Escribir pruebas para la HU 3	1

Tabla 3. 3 Tareas de programación de la HU 3.

No	HU	Iteración	Tareas de programación		
			No	Nombre	Puntos estimados
4	Obtener decisión de autorización.	2	11	Implementar la funcionalidad obtener estado de la decisión autorización	2
			12	Escribir pruebas para la HU 4	1

Tabla 3. 4 Tareas de programación de la HU 4.

No	HU	Iteración	Tareas de programación		
			No	Nombre	Puntos

					estimados
5	Probar conexión al PDP	3	13	Implementar la clase ConexionIS	0.5
			14	Escribir pruebas para la HU 4	0.5

Tabla 3. 5 Tareas de programación de la HU 5.

3.2.1 Iteración 1

Tarea de Programación	
Número de la tarea: 1	**Número de la HU:** 1
Nombre de la tarea: Crear estructura inicial del componente usando las herramientas propuestas.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Programador(es) responsable(s): Arleis Prieto	
Descripción: Mediante el desarrollo de esta tarea se crean las condiciones de trabajo para la implementación del componente. Se debe integrar el framework Carbon Studio con el IDE Eclipse. Luego comprobar que se ejecute correctamente. También se debe desplegar el Identity Server.	

Tabla 3. 6 Tarea de Programación 1 Iteración 1.

Tarea de Programación	
Número de la tarea: 2	**Número de la HU:** 1

Nombre de la tarea: Implementar la clase Fichero.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a implementar la clase Fichero con los atributos y métodos necesarios para la escritura y la lectura en el fichero.	

Tabla 3. 7 Tarea de Programación 2 Iteración 1.

Tarea de Programación	
Número de la tarea: 3	**Número de la HU:** 1
Nombre de la tarea: Diseñar interfaz visual para la configuración de los datos de conexión al PDP.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Programador(es) responsable(s): Arleis Prieto	
Descripción: Esta tarea tiene como objetivo diseñar el formulario con los campos necesarios para la entrada de los datos de configuración de la conexión la PDP.	

Tabla 3. 8 Tarea de Programación 3 Iteración 1.

Tarea de Programación	
Número de la tarea: 4	**Número de la HU:** 1
Nombre de la tarea: Escribir pruebas para la HU 1.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Programador(es) responsable(s): Arleis Prieto	

Descripción: Tarea dirigida a escribir las pruebas de aceptación de la HU 1.

Tabla 3. 9 Tarea de Programación 4 Iteración 1.

Tarea de Programación	
Número de la tarea: 5	**Número de la HU:** 2
Nombre de la tarea: Generación del cliente del servicio AuthenticationAdmin.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a generar el cliente del servicio AuthenticationAdmin a partir del wsdl del mismo.	

Tabla 3. 10 Tarea de Programación 5 Iteración 1.

Tarea de Programación	
Número de la tarea: 6	**Número de la HU:** 1
Nombre de la tarea: Implementar la funcionalidad autenticar usuario.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a implementar los atributos y métodos necesarios para la autenticación de usuario. Se debe consumir el servicio AuthenticationAdmin expuesto en el PDP.	

Tabla 3. 11 Tarea de Programación 6 Iteración 1.

Tarea de Programación

Número de la tarea: 7	**Número de la HU:** 2
Nombre de la tarea: Escribir pruebas para la HU 2.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a escribir las pruebas de aceptación de la HU 2.	

Tabla 3. 12 Tarea de Programación 7 Iteración 1.

3.2.2 Iteración 2

Tarea de Programación	
Número de la tarea: 8	**Número de la HU:** 3
Nombre de la tarea: Generación del cliente del servicio EntitlementService.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a generar el cliente del servicio EntitlementService a partir del wsdl del mismo.	

Tabla 3. 13 Tarea de Programación 8 Iteración 2.

Tarea de Programación	
Número de la tarea: 9	**Número de la HU:** 3
Nombre de la tarea: Implementar la funcionalidad realizar solicitud de autorización.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	

Descripción: Tarea dirigida a implementar los atributos y métodos necesarios para la realización de una solicitud de autorización a un recurso. Se debe consumir el servicio EntitlementService expuesto en el PDP.

Tabla 3. 14 Tarea de Programación 9 Iteración 2.

Tarea de Programación	
Número de la tarea: 10	**Número de la HU:** 3
Nombre de la tarea: Escribir pruebas para la HU 3.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a escribir las pruebas de aceptación de la HU 3.	

Tabla 3. 15 Tarea de Programación 10 Iteración 2.

Tarea de Programación	
Número de la tarea: 11	**Número de la HU:** 4
Nombre de la tarea Implementar la funcionalidad obtener estado de la decisión autorización.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 2
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a implementar los atributos y métodos necesarios para la obtención del estado de la decisión de autorización a un recurso. Se debe consumir el servicio EntitlementService expuesto en el PDP.	

Tabla 3. 16 Tarea de Programación 11 Iteración 2.

Tarea de Programación	
Número de la tarea: 12	**Número de la HU:** 4
Nombre de la tarea: Escribir pruebas para la HU 4.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a escribir las pruebas de aceptación de la HU 4.	

Tabla 3. 17 Tarea de Programación 12 Iteración 2.

3.2.3 Iteración 3

Tarea de Programación	
Número de la tarea: 13	**Número de la HU:** 5
Nombre de la tarea: Implementar la clase ConexionIS.	
Tipo de tarea: Desarrollo	**Puntos Estimados:** 0.5
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a implementar la clase ConexionIS con los atributos y métodos necesarios para la conexión al PDP.	

Tabla 3. 18 Tarea de Programación 13 Iteración 3.

Tarea de Programación	
Número de la tarea: 14	**Número de la HU:** 5
Nombre de la tarea: Escribir pruebas para la HU 5.	

Tipo de tarea: Desarrollo	**Puntos Estimados:** 1
Programador(es) responsable(s): Arleis Prieto	
Descripción: Tarea dirigida a escribir las pruebas de aceptación de la HU 5.	

Tabla 3. 19 Tarea de Programación 14 Iteración 3.

3.3 ESTÁNDARES DE CODIFICACIÓN

Los estándares de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código.

El usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas, y, posteriormente, se efectúan revisiones del código de rutinas, caben muchas posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Los estándares de codificación se definen por el equipo de desarrollo para lograr generalización en la programación del software. La generalización de aspectos tan simples como el trato de las mayúsculas, ayuda a eliminar conflictos de funcionalidades implementadas con nombres iguales y guían de forma clara el proceso de desarrollo. A continuación se muestran algunos de los ejemplos de estándares que se tuvo en cuenta para el desarrollo del componente.

Declaraciones generales:

Las declaraciones se realizan de manera descriptiva, evitando las abreviaturas y los nombres cortos.

Declaraciones de Clases:

Los nombres de las clases definidas comienzan con mayúscula al inicio de la palabra y en caso de estar conformada por palabras compuestas, la definición debe ser continua y cada palabra debe iniciar con mayúscula siguiendo el estilo determinado.

Ejemplo:

class ClienteEntitlement ;

Declaraciones de Funciones:

Las funciones deben empezar con minúscula. En caso de estar conformados por palabras compuestas, la definición debe ser continua y exceptuando la primera, cada palabra debe iniciar con mayúscula siguiendo el estilo determinado.

Ejemplo:

boolean autenticarUsuario(String usuario);

Declaraciones de Variables:

Las variables comenzarán con minúsculas, aquellas que sean compuestas se escriben de manera seguida y con las primeras letras de cada palabra en mayúsculas sin incluir la primera. En caso de que las palabras sean muy extensas se utilizan abreviaturas en inglés. Se debe declarar cada variable en una línea distinta.

Ejemplos:

boolean isAuthenticate;

authStub;

Comentarios:

Se utilizan para especificar funciones y esclarecer alguna operación específica de algún método.

La indentación correcta según Sun es:

/**

* Comentario sobre la clase A

*/

public class A { ... }

Otras normas que se siguen en el proyecto es que no se documentará lo que sea evidente.

Sentencias simples:

Cada línea debe contener una sola sentencia.

Ejemplo:

```
decision=getEstado(stub.getDecisionByAttributes(getDecision).get_return());

result = response.getFirstChildWithName(new QName("Result"));
```

Sentencias compuestas:

Deben estar indentadas a un nivel superior que el precedente. Todas las sentencias del tipo *if, for, while, do... while* deben tener llaves, incluso las que tengan una sola línea en su interior, de esta forma se evita la introducción accidental de errores si se añaden posteriormente sentencias. Debe existir un espacio entre la declaración de una sentencia y la siguiente. Las llaves deben ocupar una línea de código.

Ejemplos:

```
if (condición)

.....

...

for (inicialización, condición, actualización)
   {
          if (condición)
           {
                 ……..
           }
   }
```

3.4 FASE DE PRODUCCIÓN

Esta es la fase donde se realizan las pruebas al sistema propuesto antes de presentarse al cliente, además de tomarse decisiones sobre si incluir o no nuevas funcionalidades a la versión a presentar por cambios ocurridos durante la fase anterior. Todas las ideas y sugerencias han de ser documentadas para su posterior implementación.

3.4.1 Pruebas

Uno de los pilares fundamentales de XP es el proceso de pruebas, el cual anima a los desarrolladores a probar constantemente tanto como sea posible. Mediante esta filosofía se reduce el número de errores no detectados así como el tiempo entre la introducción de éste en el sistema y su detección. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores a la hora de introducir cambios o modificaciones.

La metodología XP utiliza dos tipos de pruebas fundamentales, las pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada.

3.4.1.1 Pruebas de aceptación

Las pruebas de aceptación son creadas en base a las HU, en cada ciclo de la iteración del desarrollo. Se debe especificar uno o diversos escenarios de prueba de aceptación para comprobar que una HU ha sido correctamente implementada. Las pruebas de aceptación son pruebas de caja negra ejecutadas por el cliente o el equipo de desarrollo, con el objetivo de comprobar que la solución implementada cumple con las funcionalidades descritas y descartar los posibles errores. Una HU puede tener todas las pruebas de aceptación que necesite para asegurar su correcto funcionamiento. El objetivo final de estas pruebas es garantizar que los requisitos han sido cumplidos y que el sistema es aceptable.[30]

Estas pruebas son de mucha importancia, ya que miden el nivel de satisfacción del cliente con cada iteración concluida, además de que marcan el final de esta y el comienzo de la próxima. A continuación se muestran una serie de casos de prueba, los cuales servirán como muestra visual del proceso de pruebas realizadas a la solución propuesta.

Dichos casos de pruebas se describirán en tablas que contendrán los siguientes campos:

- **Código:** Identificador de la prueba realizada, a su vez será sugerente al nombre de la prueba a la que hace referencia.

- **Número de HU:** Nombre de la HU a la que hace referencia la prueba a realizar.
- **Descripción de la prueba:** Breve descripción de la prueba.
- **Persona que realiza la prueba:** Nombre de la persona que realiza la prueba.
- **Condiciones de Ejecución:** Muestra las condiciones que deben cumplirse para poder llevar a cabo el caso de prueba, estas condiciones deben ser satisfechas antes de la ejecución del caso de prueba para que se puedan obtener los resultados esperados.
- **Entradas / Pasos de Ejecución:** Descripción de cada uno de los pasos seguidos durante el desarrollo de la prueba, se tendrá en cuenta cada una de las entradas que hace el usuario con el objetivo de ver si se obtiene el resultado esperado.
- **Resultados de la prueba:** Descripción breve del resultado que se espera obtener con la prueba realizada.

La evaluación de la prueba realizada se hará según el resultado de la misma, la tendrá uno de los tres resultados que a continuación se describen:

- **Satisfactoria:** Cuando el resultado de la prueba es exactamente el esperado por el usuario.
- **Parcialmente bien:** Cuando el resultado no es completamente el esperado por el cliente o usuario de la aplicación y muestra resultados erróneos o fuera de contexto.
- **Mal:** Cuando el resultado de la prueba realizada genera un error de codificación en la aplicación o muestra como resultado elementos no deseados o fuera de contexto, trayendo como consecuencia que la funcionalidad requerida por el cliente no tenga resultado, lo que invalida también la HU.

Caso de prueba de aceptación.	
Código: HU1_P1	**Número de la HU:** 1
Descripción de la prueba: Prueba para funcionalidad mostrar interfaz visual para la entrada de los datos de conexión al PDP.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path de la aplicación que lo esté usando.	

Entrada / Pasos de ejecución: El usuario hace doble clic encima del jar donde esta empaquetado el componente.
Resultado esperado: Se muestra la interfaz visual.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 3. 20 Caso de prueba de aceptación 1 Iteración 1.

Caso de prueba de aceptación.	
Código: HU1_P2	**Número de la HU:** 1
Descripción de la prueba: Prueba para funcionalidad guardar en el fichero de configuración los datos de conexión al PDP.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path de la aplicación que lo esté usando.	
Entrada / Pasos de ejecución: El usuario hace doble clic encima del jar donde esta empaquetado el componente, entra los datos correspondientes y hace clic en el botón guardar.	
Resultado esperado: Se guardan en el fichero de configuración los datos proporcionados por el usuario.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3. 21 Caso de prueba de aceptación 2 Iteración 1.

Caso de prueba de aceptación.	
Código: HU2_P1	**Número de la HU:** 2
Descripción de la prueba: Prueba para funcionalidad autenticar usuario contra el PDP.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path de la aplicación que lo esté usando. El Identity Server tiene que estar funcionando. Deben haberse guardado correctamente en el fichero los datos de conexión al Identity Server.	
Entrada / Pasos de ejecución: Un determinado usuario se intenta autenticar frente al Identity Server, el componente carga del fichero de configuración el nombre de usuario, la contraseña del mismo y la URL donde se encuentra desplegado el Identity Server, pasándole esos parámetros a la función autenticar usuario.	
Resultado esperado: Se espera como resultado la autenticación correcta del usuario en cuestión.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3. 22 Caso de prueba de aceptación 3 Iteración 1.

Caso de prueba de aceptación.	
Código: HU3_P1	**Número de la HU:** 3
Descripción de la prueba: Prueba para funcionalidad realizar solicitud de autorización a un recurso.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path	

de la aplicación que lo esté usando. El Identity Server tiene que estar funcionando y haber almacenada en esta herramienta una o varias políticas de autorización XACML. Deben haberse guardado correctamente en el fichero los datos de conexión al Identity Server.
Entrada / Pasos de ejecución: Un determinado usuario realiza la petición de autorización a un determinado recurso, pasándole a la función correspondiente el nombre de usuario, la dirección URL del recurso al que intenta acceder y la acción que va a realizar sobre el mismo.
Resultado esperado: Se espera como resultado correcta realización de la petición de autorización a un recurso.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 3. 23 Caso de prueba de aceptación 4 Iteración 1.

Caso de prueba de aceptación.	
Código: HU4_P1	**Número de la HU:** 4
Descripción de la prueba: Prueba para funcionalidad obtener decisión de autorización.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path de la aplicación que lo esté usando. El Identity Server tiene que estar funcionando y haber almacenada en esta herramienta una o varias políticas de autorización XACML. Deben haberse guardado correctamente en el fichero los datos de conexión al Identity Server. El usuario tiene que haber sido previamente autenticado.	
Entrada / Pasos de ejecución: El componente envía al PDP una petición de autorización a un determinado recurso, obteniendo así la decisión de autorización correspondiente a la solicitud realizada.	

Resultado esperado: Se espera como resultado correcta obtención de la decisión de autorización.
Evaluación de la prueba: Prueba satisfactoria.

Tabla 3. 24 Caso de prueba de aceptación 5 Iteración 2.

Caso de prueba de aceptación.	
Código: HU5_P1	**Número de la HU:** 5
Descripción de la prueba: Prueba para funcionalidad probar conexión al PDP.	
Persona que realiza la prueba: Arleis Prieto	
Condiciones de ejecución: El componente tiene que haber sido añadido al build path de la aplicación que lo esté usando. El Identity Server tiene que estar funcionando.	
Entrada / Pasos de ejecución: El usuario instancia la clase ConexionIS, modifica los atributos correspondientes e invoca la método probar conexión.	
Resultado esperado: Se espera como resultado la verificación de la conexión al Identity Server.	
Evaluación de la prueba: Prueba satisfactoria.	

Tabla 3. 25 Caso de prueba de aceptación 6 Iteración 3.

3.4.1.2 Ejecución de los casos de prueba de aceptación

El proceso de pruebas a cualquier software se realiza a través de iteraciones, donde, a medida que se procede con una nueva iteración deben haberse erradicado los defectos encontrados en la anterior, para garantizar que al final del proceso el producto quedará libre de la mayor cantidad de errores posible y listo para entregar al cliente.

Durante la ejecución de los casos de pruebas de aceptación algunas no arrojaron los resultados esperados. A continuación se listan los casos de prueba de aceptación que arrojaron alguna no conformidad durante la ejecución de la primera iteración de pruebas.

1. Mostrar interfaz visual para la entrada de los datos de conexión al PDP.
2. Guardar en el fichero de configuración los datos de conexión al PDP.

La plantilla de no conformidades constituye un registro de los defectos y fallos encontrados en el transcurso de las pruebas, cuyo principal objetivo es verificar en un futuro que estos errores fueron erradicados en posteriores iteraciones.

La siguiente tabla muestra un resumen del registro de defectos y dificultades encontradas durante la primera iteración de pruebas.

Elemento	No	No conformidad	Etapa de detección	Respuesta del desarrollador
Ventana de configuración del PEP	1	Se intentó desplegar la interfaz visual y la misma no se mostró.	Prueba de aceptación	Resuelta una vez terminada la primera iteración de pruebas
Ventana de configuración del PEP	2	Se intento guardar en el fichero de configuración los datos de conexión al PDP y los mismos no se guardaron correctamente	Prueba de aceptación	Resuelta una vez terminada la primera iteración de pruebas

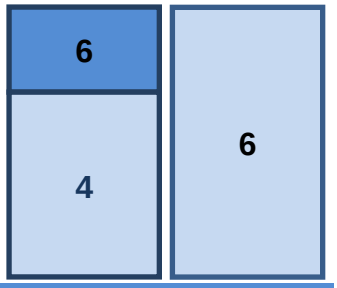
Tabla 3. 26 Resumen de defectos y dificultades.

3.4.1.3 Resultados de las pruebas de aceptación

Para que un proceso de pruebas tenga éxito se requiere de un análisis final de los resultados arrojados, es decir, la evaluación del producto que se está probando de acuerdo a todos los defectos y fallos del sistema encontrados a lo largo del proceso.

Las pruebas de aceptación se realizaron en dos iteraciones y se utilizaron los seis casos de pruebas diseñados, para verificar que las funcionalidades implementadas fueron las acordadas en las HU y que respondían correctamente a sus necesidades.

A continuación se muestra un cuadro resumen que muestra los resultados obtenidos en las pruebas ejecutadas durante las dos iteraciones realizadas.



Casos de prueba diseñados	6	6
Casos de prueba ejecutados	6	6
Casos de prueba exitosos	4	6

Figura 3. 1 Resumen de los casos de pruebas y sus resultados

Como se muestra en la figura anterior, de los 6 casos de pruebas diseñados 4 resultaron exitosos, existiendo 2 que lanzaron una no conformidad. Por tal motivo, una vez resueltos los defectos detectados se pasó a una segunda iteración de pruebas, donde se volvieron a realizar todos los casos de prueba diseñados, para verificar que fueron resueltas las no conformidades de la primera iteración y que además no habían sufrido cambios los casos de pruebas exitosos de la primera iteración. Al concluir la segunda iteración, todos los casos de prueba arrojaron resultados satisfactorios.

3.4.1.4 Pruebas de rendimiento

Las pruebas de rendimiento son aquellas que son realizadas para determinar qué tan rápido un sistema realiza una tarea bajo ciertas condiciones pre-planificadas de trabajo. Estas también son utilizadas para validar y verificar diferentes aspectos de la calidad de software, como por ejemplo, escalabilidad, fiabilidad y el buen uso de los recursos.

El uso de herramientas que permitan el monitoreo y diagnóstico del producto es de gran importancia para realizar tareas de este tipo. En este trabajo se propone el uso del Apache JMeter.

JMeter es una herramienta Java desarrollada dentro del proyecto Jakarta, que permite realizar pruebas de rendimiento y pruebas funcionales sobre aplicaciones web. JMeter

permite realizar pruebas web clásicas, pero también permite realizar test de FTP, JDBC, LDAP y servicios web. Permite la ejecución de pruebas distribuidas entre distintos ordenadores, para realizar pruebas de rendimiento. Utilizar JMeter en aplicaciones web para la comprobación de los recursos del sistema, supone una mayor efectividad en el proceso y en la fiabilidad de los resultados. JMeter como herramienta de prueba dispone de varios componentes que facilitan la elaboración de los escenarios de prueba con la ventaja de simular para cada uno de esos escenarios miles de usuarios.[32]

Con el objetivo de establecer una comparación en cuanto a rendimiento, se propone como caso de estudio una aplicación web. En un primer momento dicha aplicación gestiona el acceso a los recursos que ofrece con un mecanismo propietario, cuya implementación está en conjunto con la de la lógica del negocio. A continuación se muestran los resultados de la prueba de rendimiento realizada a la aplicación en cuestión.

Ambiente de pruebas:

Motherboard: Intel (R)

Procesador: Dual core, 2.63 GHz y 2.72 GHz

Ram: 4 Gb

Sistema operativo: Windows 7

Label	# Muestras	Media	Mediana	Mínimo	Máximo	% Error	Rendimiento (seg)	kb/ seg
/WebApplication Prueba	5	767	760	446	1068	0,00%	2,9	11,9
/WebApplication Prueba	10	63	7	4	288	0,00%	6,2	32,8
/WebApplication Prueba	5	407	455	201	589	0,00%	3,1	12,1
Total	20	325	215	4	1068	0,00%	10,2	31,7

Tabla 3. 27 Resultados de la prueba de rendimiento 1.

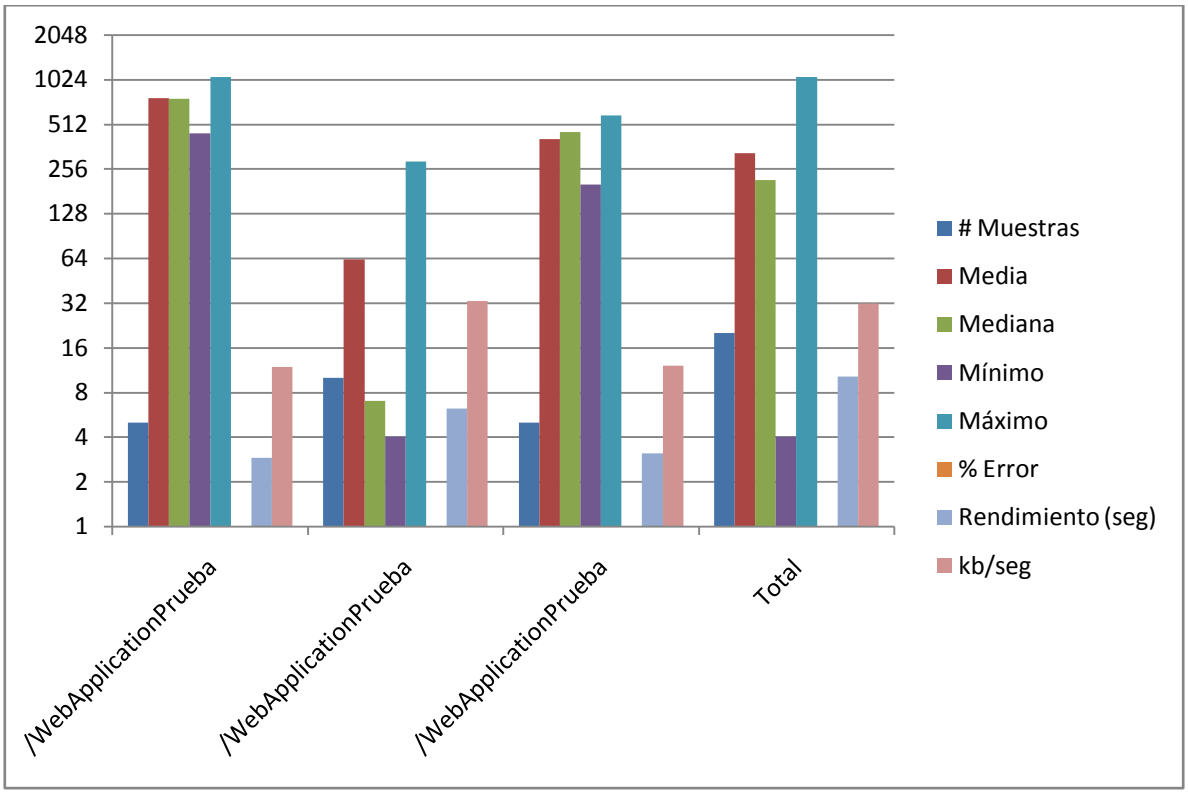


Figura 3. 2 Gráfico de resultados de la prueba de rendimiento 1.

Como se evidencia según los resultados anteriores, para 5 usuarios concurrentes el rendimiento del sistema es 2,9 segundos, para un total de 20 muestras el rendimiento del sistema alcanza los 10,2 segundos.

A continuación se muestran los resultados de la prueba de rendimiento realizada al mismo sistema y bajo las mismas condiciones, con la particularidad de que ahora dicha aplicación estará gestionando el acceso a sus recursos haciendo uso del componente de software implementado. De esta manera el sistema en cuestión externaliza el servicio de autorización.

Label	# Muestras	Media	Mediana	Mínimo	Máximo	% Error	Rendimiento (seg)	kb/ seg
/ProbandoComponente	5	50	28	10	110	0,00%	3,8	24,8
/ProbandoComponente	10	7	6	2	21	0,00%	7,9	88
/ProbandoComp	5	14	13	6	22	0,00%	4,1	30,1

onente								
Total	20	17	10	2	110	0,00%	12,3	119

Tabla 3. 28 Resultados de la prueba de rendimiento 2.

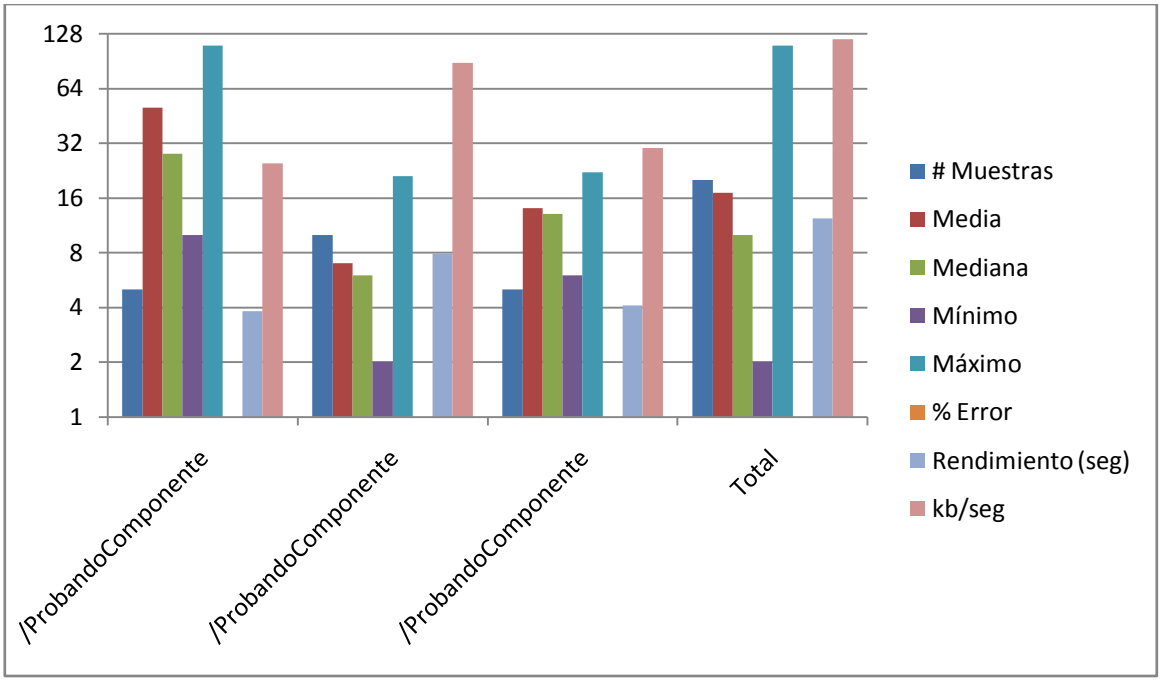


Figura 3. 3 Gráfico de resultados de la prueba de rendimiento 2.

Según los resultados anteriores, se puede apreciar que para una concurrencia de 5 usuarios el sistema responde en 3,8 segundos y para un total de 20 muestras el rendimiento del sistema alcanza los 12,3 segundos.

En el segundo caso queda de manifiesto que el rendimiento del sistema baja, debido a que incluye una consulta adicional al servidor donde se almacenan las políticas de autorización. A pesar de esto, el tiempo de rendimiento se mantienen dentro de un rango aceptable, debido a que solo hay una diferencia de entre 1 y 2 segundos con respecto al primer caso.

De todo lo anteriormente expuesto se puede concluir que: Si bien el uso del componente desarrollado no mejora el rendimiento de los sistemas con respecto al manejo de la autorización en el enfoque tradicional, el rendimiento de la aplicación se ve mínimamente afectado, debido a que la diferencia entre un caso y otro es de unos pocos segundos solamente.

3.5 CONCLUSIONES

Durante la elaboración de este capítulo se hace referencia a las fases de Implementación y Pruebas propias de la metodología de desarrollo utilizada. Durante la Fase de Implementación se describieron las tareas de programación en correspondencia con cada una de las HU identificadas. También se hace una descripción de los estándares de codificación empleados en el desarrollo del sistema. Durante la Fase de Pruebas se realizaron las pruebas de aceptación y rendimiento, garantizando que los requisitos fueron cumplidos y que el sistema es aceptable. Por último se generaron los artefactos de los casos de pruebas de aceptación.

CONCLUSIONES GENERALES

Con la culminación del presente trabajo se puede concluir que:

- El estudio del estado del arte y la elaboración del marco teórico referencial sobre el tema permitió adquirir valiosos conocimientos que posteriormente se aplicaron en el desarrollo de la solución.
- Se implementó un componente de seguridad capaz de estandarizar y externalizar el proceso de autorización en las aplicaciones informáticas desarrolladas en Java.
- La realización de pruebas al componente permitió validar la solución implementada, así como incrementar la calidad del producto.

Por todo lo antes mencionado se evidencia el cumplimiento de los objetivos propuestos en el presente trabajo de diploma, lo cual conlleva a un cumplimiento del objetivo general.

RECOMENDACIONES

Tomando como base la investigación realizada y los resultados obtenidos durante la realización de este trabajo, se recomienda lo siguiente:

- Promover el uso del estándar XACML para definir políticas de autorización en las aplicaciones desarrolladas.
- Implementación del componente en otros lenguajes de programación.
- Utilizar el componente en escenarios más complejos, donde haya un mayor número de aplicaciones que centralicen su seguridad.

REFERENCIAS BIBLIOGRÁFICAS

1. Javier, H. Aplicaciones Compuestas. 2011.

2. ¿What Are Composite Applications? [En línea] 2006 [Citado el: 3 de Diciembre del 2011] http://msdn.microsoft.com/en-us/library/bb220803.aspx.

3. Oracle, D.t.d. Gestión de aplicaciones compuestas: cómo salvar la brecha de visibilidad de IT en aplicaciones compuestas complejas. 2008.

4. Ruiz, J.G. Integración de aplicaciones de seguridad. 2008.

5. Galindo, J. Seguridad en Aplicativos SOA, Retos y Estrategias. 2009.

6. Ramarao Kanneganti, P.C. Soa Securiry. 2007: Manning Publications Co. 511.

7. Maribel Ariza Rojas, J.C.M.G. [En línea] 2009 [Citado el: 6 de Diciembre del 2011] http://pegasus.javeriana.edu.co/~jcpymes/Docs/DSBC.pdf.

8. Esponda, S. Sistema de Autorización para Web Services basado en XACML. 2007, Universidad de Belgrano.

9. Recommendation, W.C., The Platform for Privacy Preferences 1.0 (P3P 1.0) Specification. 2009.

10. Ashley, P. Hada,Satoshi, Karjoth Günter, Powers Calvin y Schunter, Matthias. Enterprise Privacy Authorization Language (EPAL 1.2). 2009.

11. Verma Manis, I. XML Security: Control information access with XACML. 2008.

12. Godik Simon, M.T. eXtensible Access Control Markup Language (XACML) Version 2.0. 2006.

13. Yared Keleta, J.H.P.E., H.S Venter. Proposing a Secure XACML architecture ensuring privacy and trust. 2009.

14. Sun's XACML Implementation, Programmer's Guide for Version 1.2. 2004.

15. Anne Anderson, H.L., SAML 2.0 profile of XACML. 2004.

16. Kaushik, N. [En línea] 2010 [Citado el: 15 de Enero del 2012] http://www.slideshare.net/OracleIDM/sans-institute-product-review-oracle-entitlements-server.

17. Hendelsen J. [En línea] 2010 [Citado el: 25 de Enero del 2012] http://www.axiomatics.com/news/20-axiomatics-announces-the-release-of-axiomatics-policy-server-40.html.

18. Readshaw, N. Tivoli Security Policy Manager. [En línea] 2009 [Citado el: 30 de Enero del 2012] http://www-01.ibm.com/software/tivoli/products/security-policy-mgr/.

19. WSO2's vision rests on four pillars of innovation. [En línea] 2010 [Citado el: 2 de Febrero del 2012] http://wso2.com/about/vision/.

20. Características del lenguaje Java. [En línea] 2009 [Citado el: 7 de Febrero del 2012] http://www.iec.csic.es/criptonomicon/java/quesjava.html.

21. Programación en Java. [En línea] 2008 [Citado el: 13 de Febrero del 2012] http://www.lenguajes-de-programacion.com/programacion-java.shtml.

22. Programas de desarrollo libres. [En línea] 2010 [Citado el: 15 de Febrero del 2012] http://www.kubuntu-es.org/wiki/desarrollo-programacion/programas-desarrollo-libres.

23. Booch, G., Rumbaugh, James y Jacobson, Ivar. El Lenguaje Unificado de Modelado. Addison Wesley. 2007.

24. Larrman, C. UML y Patrones. Introducción al análisis y diseño orientado a objetos. 2001.

25. visual-paradigm. [En línea] 2008 [Citado el: 16 de Febrero del 2012] http://www.visual-paradigm.com.

26. Coimbra, V.H.G. Modelos de procesos de desarrollo, ciclo de vida de desarrollo software, MSF, MOF, ITIL. 2009.

27. Méndez, A.V. Metodologías de Desarrollo de Software. 2010.

28. Patricio Letelier, C.P. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). 2009.

29. Bonifaz, V. El diagnóstico y las organizaciones. [En línea] 2009 [Citado el: 2 de Marzo del 2012] http://www.mktglobal.iteso.mx/index.php?option=com_content&view=article&id=407&Itemid=125.

30. Kent Beck, M.F., Planning Extreme Programming. Addison Wesley. 2000.

31. Kniberg, H., Scrum and XP from the Trenches. 2008.

32. Almenares, L.S., Cómo realizar Pruebas de Carga y Estrés en JMeter. 2009.

ANEXOS

Anexo 1: Reconocimiento internacional otorgado al Identity Server en el 2011.

Figura 1 Premio a mejor solución informática en ofrecer implementaciones de XACML y OpenID.

Anexo 2: Pruebas de carga realizadas al Identity Server.

Ambiente de las pruebas:

- Intel(R) Xeon(R) CPU X3440 @ 2.53GHz procesador, 4 GB RAM, SO – Debian 6.0 (64bit) – con una sola instancia de Identity Server.
- [-Xms1024m -Xmx2024m -XX:MaxPermSize=1024m].
- Complejidad de las políticas.

  - L1: 10 reglas por política, teniendo cada regla 1 atributo.
  - L2: 100 reglas por política, teniendo cada regla más de 10 atributos.

- Peticiones

  - Un millón de peticiones XACML.
  - Peticiones XACML recuperadas aleatoriamente a partir de un grupo de 10 000 peticiones diferentes disponibles.

En la siguiente imagen se pueden ver los resultados de una prueba de carga, con una concurrencia de 100 0 200 usuarios haciendo peticiones al Identity Server con diferentes niveles de complejidad de las políticas a evaluar. El IS procesa 1 millón de peticiones y se pueden ver los resultados, en la penúltima columna sin usar el mecanismo de cacheo y en la última columna usando el mecanismo de cacheo. La mejora en el rendimiento es notable.

Número de políticas	Complejidad	Concurrencia	Rendimiento sin cacheo (tps)	Rendimiento con cacheo (tps)
100	L1	100	3956	16008
100	L1	200	3864	16370
100	L2	100	3391	15746
100	L2	200	3258	15786
1000	L1	100	2591	16679
1000	L1	200	2477	16938

1000	L2	100	1101	16130
1000	L2	200	1104	16411

Tabla 1 Resultados de prueba de carga al Identity Server usando Cahing.

tps: Transacciones por segundo.

En la segunda tabla se hizo la misma prueba pero usando otro mecanismo para mejorar el rendimiento llamado Thrift (Ahorro). A continuación se muestra como los resultados se mantienen.

Número de políticas	**Complejidad**	**Concurrencia**	**Rendimiento con HTTP (tps)**	**Rendimiento con Thrift (tps)**
100	L1	100	977	16008
100	L1	200	972	16370
100	L2	100	977	15746
100	L2	200	973	15786
1000	L1	100	976	16679
1000	L1	200	970	16938
1000	L2	100	971	16130
1000	L2	200	977	16411

Tabla 2 Resultados de prueba de carga al Identity Server usando Thrift.

tps: Transacciones por segundo.

Anexo 3: Diagrama de clases.

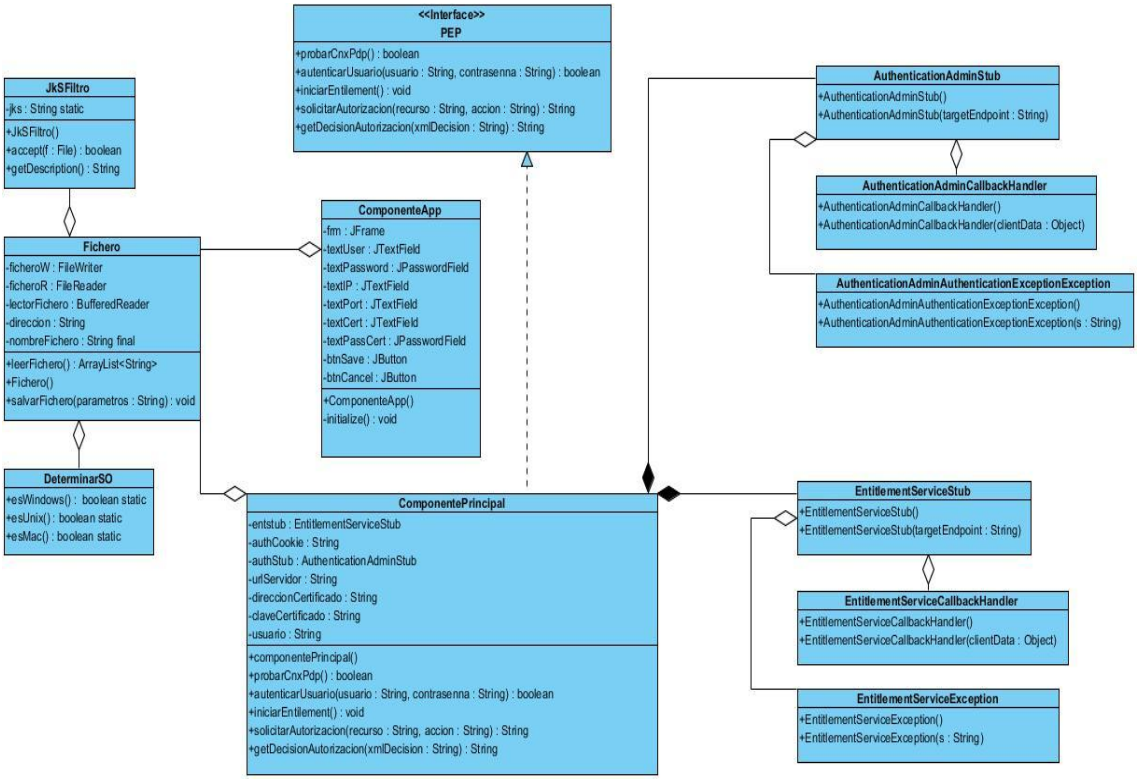


Figura 2 Diagrama de clases.